THE UNIVERSITY
of EDINBURGH

University of Edinburgh

School of Informatics

Doctor of Philosophy

Neural Density Estimation and Likelihood-free Inference

by

George Papamakarios

April 2019

Abstract

I consider two problems in machine learning and statistics: the problem of estimating the joint probability density of a collection of random variables, known as *density estimation*, and the problem of inferring model parameters when their likelihood is intractable, known as *likelihood-free inference*. The contribution of the thesis is a set of new methods for addressing these problems that are based on recent advances in neural networks and deep learning.

The first part of the thesis is about density estimation. The joint probability density of a collection of random variables is a useful mathematical description of their statistical properties, but can be hard to estimate from data, especially when the number of random variables is large. Traditional density-estimation methods such as histograms or kernel density estimators are effective for a small number of random variables, but scale badly as the number increases. In contrast, models for density estimation based on neural networks scale better with the number of random variables, and can incorporate domain knowledge in their design. My main contribution is *Masked Autoregressive Flow*, a new model for density estimation based on a bijective neural network that transforms random noise to data. At the time of its introduction, Masked Autoregressive Flow achieved state-of-the-art results in general-purpose density estimation. Since its publication, Masked Autoregressive Flow has contributed to the broader understanding of neural density estimation, and has influenced subsequent developments in the field.

The second part of the thesis is about likelihood-free inference. Typically, a statistical model can be specified either as a likelihood function that describes the statistical relationship between model parameters and data, or as a simulator that can be run forward to generate data. Specifying a statistical model as a simulator can offer greater modelling flexibility and can produce more interpretable models, but can also make inference of model parameters harder, as the likelihood of the parameters may no longer be tractable. Traditional techniques for likelihood-free inference such as approximate Bayesian computation rely on simulating data from the model, but often require a large number of simulations to produce accurate results. In this thesis, I cast the problem of likelihood-free inference as a density-estimation problem, and address it with neural density models. My main contribution is the introduction of two new methods for likelihood-free inference: *Sequential Neural Posterior Estimation (Type A)*, which estimates the posterior, and *Sequential Neural Likelihood*, which estimates the likelihood. Both methods use a neural density model to estimate the posterior/likelihood, and a sequential training procedure to guide simulations. My experiments show that the proposed methods produce accurate results, and are often orders of magnitude faster than alternative methods based on approximate Bayesian computation.

Lay summary

A big part of science is about understanding how various quantities of interest relate to each other. For example, a weather scientist may want to know how the temperature today relates to the temperature tomorrow. In practice, we often want to answer questions such as "if the temperature today is high, how likely is it that the temperature tomorrow will be high too?". In order to answer such questions successfully, it's useful to express relationships between quantities of interest (also known as variables) using the mathematical language of probability. This thesis presents a set of new mathematical techniques for calculating probabilistic relationships between variables more effectively.

One way to calculate probabilistic relationships between variables is to first collect a lot of data on them, and then analyze the data to measure the relationships between the variables directly. For example, we may measure the temperature over a number of days, and then analyze these measurements to calculate how the various temperatures relate to each other. This is easy for a small number of days, but for a large number of days it would be hard to calculate the relationships between all possible combinations of temperatures. The first part of the thesis describes techniques for calculating probabilistic relationships from data even when the number of variables grows very large.

Another way to estimate probabilistic relationships is to use our knowledge of the physical world in order to create a computer simulation of how some variables affect other variables. For example, a weather expert might know the explicit mechanism of how the temperature today will affect the temperature tomorrow. However, in such situations it may still be hard to know how variables relate in the opposite direction. For example, if we know that the temperature today is high, how can we know how likely it is that the temperature yesterday was high too, if we can only know how temperature changes forward in time? The second part of the thesis describes techniques for calculating relationships in the backward direction when we have a computer simulation that tells us relationships only in the forward direction.

Acknowledgements

I'm grateful to Iain Murray for supervising me during my PhD. Iain has had a massive impact on my research, my thinking, and my professional development; his contributions extend far beyond this thesis. I'm also grateful to John Winn and Chris Williams, who served on my PhD committee and gave me useful feedback in our annual meetings.

I'm grateful to my co-authors Theo Pavlakou and David Sterratt for their help and contribution. Conor Durkan, Maria Gorinova and Amos Storkey generously read parts of this thesis and gave me useful feedback. I'm grateful to Michael Gutmann and Kyle Cranmer for our discussions, which shaped part of the thesis, and to Johann Brehmer for fixing an important error in my code.

I'd like to extend a special thank you to the developers of Theano (Al-Rfou et al., 2016), which I used in all my experiments. Their contribution to machine-learning research as a whole has been monumental, and they deserve significant credit for that.

I'm grateful to the Centre for Doctoral Training in Data Science, the Engineering and Physical Sciences Research Council, the University of Edinburgh and Microsoft Research, whose generous funding and support enabled me to pursue a PhD.

Declaration

I declare that this thesis was composed by me, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*George Papamakarios*)

Contents

Chapter 1

Introduction

Density estimation and *likelihood-free inference* are two fundamental problems of interest in machine learning and statistics; they lie at the core of probabilistic modelling and reasoning under uncertainty, and as such they play a significant role in scientific discovery and artificial intelligence. The goal of this thesis is to develop a new set of methods for addressing these two problems, based on recent advances in neural networks and deep learning.

The first part of the thesis is about *density estimation*, the problem of estimating the joint probability density of a collection of random variables from samples. In a sense, density estimation is the reverse of sampling: in density estimation, we are given samples and we want to retrieve the density function from which the samples were generated; in sampling, we are given a density function and we want to generate samples from it.

Density estimation addresses one of the most fundamental problems in machine learning, the problem of discovering structure from data in an unsupervised manner. A density function is a complete description of the joint statistical properties of the data, and in that sense a model of the density function can be viewed as a model of data structure. As such, a model of the density function can be used in a variety of downstream tasks that involve knowledge of data structure, such as inference, prediction, data completion and data generation.

The second part of the thesis is about *statistical inference*, the problem of inferring parameters of interest from observations given a model of their statistical relationship. In this work, I adopt a Bayesian framework for statistical inference: beliefs over unknown quantities are represented by density functions, and Bayes' rule is used to update prior beliefs given new observations. Bayesian inference is one of the two main approaches to statistical inference (the other one being frequentist inference), and is widely used in science and engineering. From now on, if not made specific, the term 'inference' will refer to 'Bayesian inference'.

Likelihood-free inference refers to the situation where the likelihood of the model is too expensive to evaluate, which is typically the case when the model is specified as a simulator that stochastically generates observations given parameters. Such simulators are used ubiquitously in science and engineering for modelling complex mechanistic processes of the real world; as a result, several scientific and engineering problems can be framed as likelihood-free inference of a simulator's parameters. The goal of likelihood-free inference is to compute posterior beliefs over parameters using simulations from the model rather than likelihood evaluations.

In this thesis, I approach both density estimation and likelihood-free inference as machine-learning problems; in either case, the task is to estimate a density model from data. The methods developed in this thesis are heavily based on neural networks and deep learning. The use of deep

learning is motivated by two reasons. First, neural networks have demonstrated state-of-the-art performance in a wide variety of machine-learning tasks, such as computer vision (Krizhevsky et al., 2012), natural-language processing (Devlin et al., 2018), generative modelling (Radford et al., 2016), and reinforcement learning (Silver et al., 2016) — as we will see in this thesis, neural networks advance the state of the art in density estimation and likelihood-free inference too. Second, deep learning is actively supported by software frameworks such as *Theano* (Al-Rfou et al., 2016), *TensorFlow* (Abadi et al., 2015) and *PyTorch* (Paszke et al., 2017), which provide access to powerful hardware (such as graphics-processing units) and facilitate designing and training neural networks in practice. This thesis focuses on the application of neural networks to density estimation and likelihood-free inference, and not on deep learning itself; knowledge of deep learning is generally assumed, but not absolutely required. A comprehensive review of neural networks and deep learning is provided by Goodfellow et al. (2016).

1.1 List of contributions

The main contribution of this thesis is a set of new methods for density estimation and likelihood-free inference, which are based on techniques from neural networks and deep learning. In particular, the new methods contributed by this thesis are the following:

(i) **Masked Autoregressive Flow**, an expressive neural density model whose density is tractable to evaluate. MAF can be trained on data to estimate their underlying density function. At the time of its introduction, MAF achieved state-of-the-art performance in density estimation, and has been influential ever since.

(ii) **Sequential Neural Posterior Estimation (Type A)**, a method for likelihood-free inference of simulator models that is based on neural density estimation. SNPE-A trains a neural density model on simulated data to approximate the posterior density of the parameters given observations. The designation 'Type A' is in order to distinguish SNPE-A from its 'Type-B' variant, which was proposed later. SNPE-A was shown to both improve accuracy and dramatically reduce the required number of simulations compared to traditional methods for likelihood-free inference.

(iii) **Sequential Neural Likelihood**, an alternative method for likelihood-free inference that uses a neural density model to approximate the likelihood instead of the posterior. SNL can be as fast and accurate as SNPE-A, but is more generally applicable and significantly more robust. Unlike SNPE-A, SNL can be used with Masked Autoregressive Flow.

1.2 Structure of the thesis

The thesis is divided into two parts: part I is about *neural density estimation*, whereas part II is about *likelihood-free inference*. Each part has a separate introduction and a separate conclusion. The two parts are intended to be standalone, and can be read independently. However, the part on likelihood-free inference makes heavy use of density-estimation techniques, hence, although not absolutely necessary, I would recommend that the part on density estimation be read first.

The thesis consists of three chapters, each of which is centred on a different published paper. Each chapter includes the paper as published; in addition, it provides extra background in order to motivate the paper, and evaluates the contribution and impact of the paper since its publication. The three chapters are listed below:

Chapter 2 is about **Masked Autoregressive Flow**, and is based on the following paper:

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems 30*, 2017

Chapter 3 is about **Sequential Neural Posterior Estimation (Type A)**, and is based on the following paper:

G. Papamakarios and I. Murray. Fast ϵ-free inference of simulation models with Bayesian conditional density estimation. *Advances in Neural Information Processing Systems 29*, 2016

Chapter 4 is about **Sequential Neural Likelihood**, and is based on the following paper:

G. Papamakarios, D. C. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019

Part I

Neural density estimation

Chapter 2

Masked Autoregressive Flow for Density Estimation

This chapter is devoted to *density estimation*, and how we can use neural networks to estimate densities. We start by explaining what density estimation is, why it is useful, and what challenges it presents (section 2.1). We then review some standard methods for density estimation, such as *mixture models*, *histograms* and *kernel density estimators*, and introduce the idea of *neural density estimation* (section 2.2). The main contribution of this chapter is the paper *Masked Autoregressive Flow of Density Estimation*, which presents *Masked Autoregressive Flow*, a new model for density estimation (sections 2.3 and 2.4). We conclude the chapter by reviewing advances in neural density estimation since the publication of the paper (section 2.5), and by discussing how neural density estimators fit more broadly in the space of generative models (section 2.6).

2.1 Density estimation: what and why?

Suppose we have a stationary process that generates data. The process could be of any kind, as long as it doesn't change over time: it could be a black-box simulator, a computer program, an agent acting, the physical world, or a thought experiment. Suppose that each time the process is run forward it independently generates a D-dimensional real vector $\mathbf{x}'$. We can think of the *probability density* at $\mathbf{x} \in \mathbb{R}^D$ as a measure of how often the process generates data near $\mathbf{x}$ per unit volume. In particular, let $B_\epsilon(\mathbf{x})$ be a ball centred at $\mathbf{x}$ with radius $\epsilon > 0$, and let $|B_\epsilon(\mathbf{x})|$ be its volume. Informally speaking, the probability density at $\mathbf{x}$ is given by:

$$p(\mathbf{x}) = \frac{\Pr(\mathbf{x}' \in B_\epsilon(\mathbf{x}))}{|B_\epsilon(\mathbf{x})|} \quad \text{for } \epsilon \to 0, \tag{2.1}$$

where $\Pr(\mathbf{x}' \in B_\epsilon(\mathbf{x}))$ is the probability that the process generates data in the ball. For brevity, I will often just say *density* and mean probability density.

A function $p(\cdot)$ that takes an arbitrary vector $\mathbf{x}$ and outputs the density at $\mathbf{x}$ is called a *probability-density function*. To formally define the density function, suppose that $\Pr(\mathbf{x}' \in \mathcal{X})$ is a probability measure defined on all Lebesgue-measurable subsets of $\mathbb{R}^D$. We require $\Pr(\mathbf{x}' \in \mathcal{X})$ to be absolutely continuous with respect to the Lebesgue measure, which means $\Pr(\mathbf{x}' \in \mathcal{X})$ is zero if $\mathcal{X}$ has zero volume. A real-valued function $p(\cdot)$ is called a probability-density function if it has the following

two properties:

$$p(\mathbf{x}) \geq 0 \quad \text{for all } \mathbf{x} \in \mathbb{R}^D, \tag{2.2}$$

$$\int_{\mathcal{X}} p(\mathbf{x})\,\mathrm{d}\mathbf{x} = \Pr(\mathbf{x}' \in \mathcal{X}) \quad \text{for all Lebesgue-measurable } \mathcal{X} \subseteq \mathbb{R}^D. \tag{2.3}$$

The second property implies that a density function must integrate to 1, that is:

$$\int_{\mathbb{R}^D} p(\mathbf{x})\,\mathrm{d}\mathbf{x} = 1. \tag{2.4}$$

The *Radon–Nikodym theorem* (Billingsley, 1995, theorem 32.2) guarantees that a density function exists and is unique almost everywhere, in the sense that any two density functions can only differ in a set of zero volume.

The density function is not defined if $\Pr(\mathbf{x}' \in \mathcal{X})$ is not absolutely continuous. For example, this is the case if $\Pr(\mathbf{x}' \in \mathcal{X})$ is discrete, i.e. concentrated on a countable subset of $\mathbb{R}^D$. From now on we will assume that the density function always exists, but many of the techniques discussed in this thesis can be adapted for e.g. discrete probability measures.

In practice, we often don't have access to the density function of the process we are interested in. Rather, we have a set of datapoints generated by the process (or the ability to generate such a dataset) and we would like to estimate the density function from the dataset. Hence, the problem of *density estimation* can be stated as follows:

> *Given a set $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ of independently and identically generated datapoints, how can we estimate the probability density at an arbitrary location $\mathbf{x}$?*

2.1.1 Why estimate densities?

Before I address the problem of *how* to estimate densities, I will discuss the issue of *why*. The question I will attempt to answer is the following:

> *Why is the density function useful, and why should we expend resources trying to estimate it?*

To begin with, a model of the density function is a complete statistical model of the generative process. With the density function, we can (up to computational limitations) do the following:

(i) Calculate the probability of any Lebesgue-measurable subset of $\mathbb{R}^D$ under the process, by integration using property (2.3).

(ii) Sample new data using general-purpose sampling algorithms, such as Markov-chain Monte Carlo (Murray, 2007; Neal, 1993).

(iii) Calculate expectations under the process using integration:

$$\mathbb{E}(f(\mathbf{x})) = \int_{\mathbb{R}^D} f(\mathbf{x})p(\mathbf{x})\,\mathrm{d}\mathbf{x}. \tag{2.5}$$

(iv) Test the density model against data generated from the actual process, using one-sample goodness-of-fit testing (Chwialkowski et al., 2016; Jitkrittum et al., 2017; Liu et al., 2016).

Nevertheless, although the above are useful applications of the density function, I would argue that they don't necessitate modelling the density function. Indeed, one could estimate a *sampling model* from the data, i.e. a model that generates data in way that is statistically similar to the process. A sampling model can be used (again up to computational limitations) instead of a density model in all the above applications as follows:

(i) The probability of a Lebesgue-measurable subset $\mathcal{X} \subset \mathbb{R}^D$ can be estimated by the fraction of samples falling in $\mathcal{X}$.

(ii) Sampling new data is trivial (and in practice usually much more efficient) with the sampling model.

(iii) Calculating expectations can be estimated by Monte-Carlo integration.

(iv) Testing the model against the actual process can be done with a two-sample test (Gretton et al., 2012).

So why invest in density models if there are other ways to model the desirable statistical properties of a generative process? In the following, I discuss some applications for which knowing the value of the density is important.

Bayesian inference

In the context of Bayesian inference, density functions encode degrees of belief. Bayes' rule describes how beliefs should change in light of new evidence as a operation over *densities*. In particular, if we express our beliefs about a quantity $\boldsymbol{\theta}$ using a density model $p(\boldsymbol{\theta})$ and the statistical relationship between $\mathbf{x}$ and $\boldsymbol{\theta}$ using a conditional density model $p(\mathbf{x} \,|\, \boldsymbol{\theta})$, we can calculate how our beliefs about $\boldsymbol{\theta}$ should change in light of observing $\mathbf{x}$ by:

$$p(\boldsymbol{\theta} \,|\, \mathbf{x}) \propto p(\mathbf{x} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta}). \tag{2.6}$$

Being able to estimate densities such as the prior $p(\boldsymbol{\theta})$, the likelihood $p(\mathbf{x} \,|\, \boldsymbol{\theta})$ and the posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x})$ from data can be valuable for Bayesian inference. For example, density estimation on large numbers of unlabelled data can be used for constructing effective priors (Zoran and Weiss, 2011). In cases where the likelihood is unavailable, density estimation on joint examples of $\mathbf{x}$ and $\boldsymbol{\theta}$ can be used to model the posterior or the likelihood (Lueckmann et al., 2017, 2018; Papamakarios and Murray, 2016; Papamakarios et al., 2019). A big part of this thesis is dedicated to using density estimation for likelihood-free inference, and this is a topic that I will examine thoroughly in the following chapters.

Data compression

There is a close relationship between density modelling and data compression. Suppose we want to encode a message $\mathbf{x}$ up to a level of precision defined by a small neighbourhood $B(\mathbf{x})$ around $\mathbf{x}$. Assuming the message was generated from a distribution with density $p(\mathbf{x})$, the information content associated with $\mathbf{x}$ at this level of precision is:

$$I(\mathbf{x}) = -\log \int_{B(\mathbf{x})} p(\mathbf{x}')\, \mathrm{d}\mathbf{x}' \approx -\log p(\mathbf{x}) - \log |B(\mathbf{x})|\,. \tag{2.7}$$

The above means that the density at $\mathbf{x}$ tells us how many bits an optimal compressor should use to encode $\mathbf{x}$ at a given level of precision (assuming base-2 logarithms). Conversely, a data compressor implicitly defines a density model. Given a perfect model of the density function, data compressors such as arithmetic coding (MacKay, 2002, section 6.2) can achieve almost perfect compression. On the other hand, if the data compressor uses (explicitly or implicitly) a density model $q(\mathbf{x})$ that is not the same as the true model $p(\mathbf{x})$, the average number of wasted bits is:

$$\mathbb{E}_{p(\mathbf{x})}(-\log q(\mathbf{x}) - \log |B(\mathbf{x})|) - \mathbb{E}_{p(\mathbf{x})}(-\log p(\mathbf{x}) - \log |B(\mathbf{x})|) = D_{\mathrm{KL}}(p(\mathbf{x}) \,\|\, q(\mathbf{x})) > 0. \tag{2.8}$$

Hence, a more accurate density model implies a more efficient data compressor, and vice versa.

Model training and evaluation

Even if we are not interested in estimating the density function per se, the density of the training data under the model is useful as an objective for training the model. For example, suppose we wish to fit a density model $q(\mathbf{x})$ to training data $\{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ using maximum-likelihood estimation. The objective to be maximized at training time is:

$$L(q) = \frac{1}{N} \sum_n \log q(\mathbf{x}_n). \tag{2.9}$$

After training, being able to calculate $L(q)$ on a held-out test set is useful for ranking and comparing models. Hence, the usefulness of the density function as a training and evaluation objective motivates endowing our models with density-estimation capabilities even if we don't intend to use the model as a density estimator per se.

It is possible to formulate training and evaluation objectives that don't explicitly require densities, for example based on likelihood ratios (Goodfellow et al., 2014), kernel-space discrepancies (Dziugaite et al., 2015), or Wasserstein distances (Arjovsky et al., 2017). One argument in favour of the maximum-likelihood objective is its good asymptotic properties: a maximum-likelihood density estimator is *consistent*, i.e. it converges in probability to the density being estimated (assuming it's sufficiently flexible to represent it), and *efficient*, i.e. among all consistent estimators it attains the lowest mean squared error (Wasserman, 2010, section 9.4). Moreover, in the limit of infinite data, maximizing $\frac{1}{N} \sum_n \log q(\mathbf{x}_n)$ is equivalent to minimizing $D_{\mathrm{KL}}(p(\mathbf{x}) \,\|\, q(\mathbf{x}))$. In addition to its interpretation as a measure of efficiency of a data compressor which I already discussed, the KL divergence is the only divergence between density functions that possesses a certain set of properties, namely *locality*, *coordinate invariance* and *subsystem independence*, as defined by Caticha (2004). Rezende (2018) has argued that these properties of the KL divergence justify its use as a training and evaluation objective, and explain its popularity in machine learning.

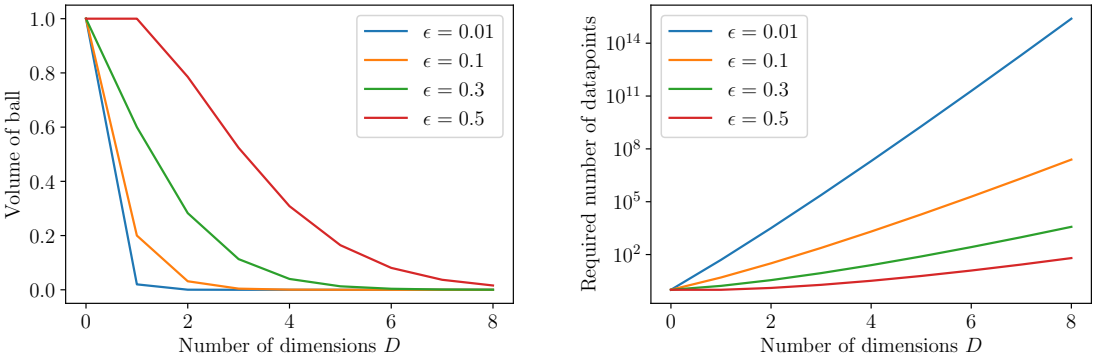
Density models as components of other algorithms

Density models are often found as components of other algorithms in machine learning and statistics. For instance, sampling algorithms such as importance sampling and sequential Monte Carlo rely on an auxiliary model, known as the *proposal*, which is required to provide the density of its samples (Gu et al., 2015; Müller et al., 2018; Paige and Wood, 2016; Papamakarios and Murray, 2015). Variational autoencoders (which will be discussed in more detail in section 2.6.2) require two density models as subcomponents, known as the *encoder* and the *prior* (Kingma and Welling, 2014; Rezende et al., 2014). Finally, in variational inference of continuous parameters, the approximate posterior is required to be a density model over the parameters of interest (Kucukelbir et al., 2015; Ranganath et al., 2014).

2.1.2 Density estimation in high dimensions

Estimating densities in high dimensions is a hard problem. Naive methods that work well in low dimensions often break down as the dimensionality increases (as I will discuss in more detail in section 2.2). The observation that density estimation, as well as other machine-learning tasks, becomes dramatically harder as the dimensionality increases is often referred to as the *curse of dimensionality* (Hastie et al., 2001, section 2.5; Bishop, 2006, section 1.4).

I will illustrate the curse of dimensionality for density estimation with a simple example. Consider a process that generates data uniformly in the D-dimensional unit cube $[0, 1]^D$. Clearly, the



(a) Volume of a D-dimensional ball of radius ϵ.

(b) Expected number of datapoints we need to generate until one falls in the ball.

Figure 2.1: Illustration of the curse of dimensionality for density estimation.

density $p(\mathbf{x})$ is equal to 1 for all $\mathbf{x}$ inside the cube. Suppose that we try to estimate $p(\mathbf{x})$ for some $\mathbf{x}$ inside the cube using the fraction of training datapoints that fall into a small ball around $\mathbf{x}$. The expected fraction of training datapoints that fall into a ball $B_\epsilon(\mathbf{x})$ centred at $\mathbf{x}$ with radius ϵ is no greater than the volume of the ball, which is given by:

$$|B_\epsilon(\mathbf{x})| = \frac{\left(\pi^{1/2}\epsilon\right)^D}{\Gamma\left(\frac{D}{2}+1\right)}, \tag{2.10}$$

where $\Gamma(\cdot)$ is the Gamma function. In practice, we would need to make the ball large enough to contain at least one datapoint, otherwise the estimated density will be zero. However, no matter how large we make the radius ϵ, the volume of the ball approaches zero as D grows larger. In other words, in a dimension high enough, almost all balls will be empty, even if we make their radius larger than the side of the cube!

Figure 2.1a illustrates the shrinkage in volume of a D-dimensional ball as D increases. Figure 2.1b shows the expected number of datapoints we would need to generate from the process until one of them falls into the ball, which is no less than the inverse of the ball's volume. As we can see, for a ball of radius $\epsilon = 0.01$ and in $D = 8$ dimensions, we would need a dataset of size at least a *thousand trillion* datapoints!

In practice, in order to scale density estimation to high dimensions without requiring astronomical amounts of data, we make assumptions about the densities we wish to estimate and encode these assumptions into our models. With careful model design, it has been possible to train good density models on high-dimensional data such as high-resolution images (Dinh et al., 2017; Kingma and Dhariwal, 2018; Menick and Kalchbrenner, 2019; Salimans et al., 2017) and audio streams (Kim et al., 2018; Prenger et al., 2018; van den Oord et al., 2016a). Fortunately, we can often make assumptions that are generic enough to apply to broad domains, while still making density estimation practical. In the following, I will examine a few generic assumptions that often guide model design.

Smoothness

We often assume that the density function varies smoothly, that is, if $\|\mathbf{x}_A - \mathbf{x}_B\|$ is small, then $|p(\mathbf{x}_A) - p(\mathbf{x}_B)|$ is also small. The assumption of smoothness encourages the model to interpolate over small regions that happen to have no training data due to sampling noise, rather than assign

zero density to them. In practice, we enforce smoothness either by limiting the flexibility of the density model, or by regularizing it.

Low intrinsic dimensionality

We often assume that the world has fewer degrees of freedom than the measurements we make to describe it. For example, an image of a natural scene could only vary in certain semantically meaningful ways (e.g. location of objects in the scene, direction of lighting, etc.), whereas each pixel of the image can't vary arbitrarily. This means that such natural images would approximately lie on a manifold of low intrinsic dimensionality embedded in the D-dimensional space of pixel intensities — see also discussions by Basri and Jacobs (2003) and Hinton et al. (1997) on the low-dimensional manifold structure of simple images. In practice, low-dimensional manifold structure can be modelled by limiting the degrees of freedom that the model can represent (e.g. by introducing information bottlenecks in the model's structure).

Symmetries and invariances

Real data often have symmetries, some examples of which are listed below.

(i) *Translation symmetry.* An image of a car moved a few pixels to the right is still an image of the same car.

(ii) *Scale symmetry.* An audio stream of music played twice as fast may still be a plausible piece of music.

(iii) *Mirror symmetry.* An image flipped horizontally may still be a plausible image.

(iv) *Order symmetry.* A dataset whose datapoints have been reordered is still the same dataset.

Such symmetries in the data are often reflected in model design. For example, top-performing image models employ convolutions and multi-scale architectures (Dinh et al., 2017; Kingma and Dhariwal, 2018; Menick and Kalchbrenner, 2019), which equip the model with a degree of translation and scale symmetry. Additionally, symmetries in the data can be used for data augmentation: for instance, an image model can be trained with all training images flipped horizontally as additional training data (Dinh et al., 2017; Papamakarios et al., 2017).

Independencies and loose dependencies

Sometimes we know or suspect that certain measurements are either independent of or at least loosely dependent on other measurements. For example, in an audio stream, one could assume that the current audio intensity is only loosely dependent on audio intensities of more than a few seconds before it. This could be encoded in a model by limiting the range (or the receptive field) of each variable in the data, for example in the case of audio (van den Oord et al., 2016a) or pixel intensities (van den Oord et al., 2016b).

2.2 Methods for density estimation

Methods for density estimation can broadly be classified as either *parametric* or *non-parametric*. Parametric methods model the density function as a specified functional form with a fixed number of tunable parameters. Non-parametric methods are those that don't fit the above description: typically, they specify a model whose complexity grows with the number of training datapoints.

In this section I will discuss and compare some standard density-estimation methods from both categories, and I will introduce the idea of *neural density estimation*.

2.2.1 Simple parametric models and mixture models

In parametric density estimation, we first specify a density model $q_{\boldsymbol{\phi}}(\mathbf{x})$ with a fixed number of tunable parameters $\boldsymbol{\phi}$, and then we try to find a setting of $\boldsymbol{\phi}$ that makes $q_{\boldsymbol{\phi}}(\mathbf{x})$ as similar as possible to the true density $p(\mathbf{x})$. A straightforward approach is to choose $q_{\boldsymbol{\phi}}(\mathbf{x})$ to be in a simple parametric family, for example the *Gaussian family*:

$$q_{\boldsymbol{\phi}}(\mathbf{x}) = \frac{1}{\left|\det(2\pi\boldsymbol{\Sigma})\right|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right) \quad \text{where} \quad \boldsymbol{\phi} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}\,. \tag{2.11}$$

The parameters of the Gaussian family are a real D-dimensional vector $\boldsymbol{\mu}$ and a $D \times D$ symmetric positive-definite matrix $\boldsymbol{\Sigma}$. There are also special cases of the Gaussian family that further restrict the form of $\boldsymbol{\Sigma}$, such as the probabilistic versions of *principal-components analysis* (Tipping and Bishop, 1999), *minor-components analysis* (Williams and Agakov, 2002), and *extreme-components analysis* (Welling et al., 2004).

The problem with simple parametric families such as the above is that the set of density functions they can represent is limited. For example, the Gaussian family can't represent density functions with more than one mode. One way to increase the expressivity of parametric models is to combine a number of models into one *mixture model*. Let $q^{(1)}_{\boldsymbol{\phi}_1}(\mathbf{x}), \ldots, q^{(K)}_{\boldsymbol{\phi}_K}(\mathbf{x})$ be K parametric models from the same or different families. A mixture model is a parametric model defined as:

$$q_{\boldsymbol{\phi}}(\mathbf{x}) = \sum_k \alpha_k \, q^{(k)}_{\boldsymbol{\phi}_k}(\mathbf{x}) \quad \text{where} \quad \sum_k \alpha_k = 1 \quad \text{and} \quad \alpha_k \geq 0 \;\; \text{for all } k. \tag{2.12}$$

The parameters of a mixture model are $\boldsymbol{\phi} = \{\alpha_1, \boldsymbol{\phi}_1, \ldots, \alpha_K, \boldsymbol{\phi}_K\}$. Mixture models where all $q^{(k)}_{\boldsymbol{\phi}_k}(\mathbf{x})$ are Gaussian are usually referred to as *Gaussian mixture models*. Gaussian mixture models are a strong density-estimation baseline: with sufficiently many components they can approximate any density arbitrarily well (McLachlan and Basford, 1988). However, they may require a large number of components to approximate density functions that can be expressed compactly in a different form (e.g. a uniform density in the unit cube would require a large number of narrow Gaussians to approximate its steep boundary).

Parametric density models are typically estimated by maximum likelihood. Given a set of training datapoints $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ that have been independently and identically generated by a process with density $p(\mathbf{x})$, we seek a setting of the model's parameters $\boldsymbol{\phi}$ that maximize the average log likelihood on the training data:

$$L(\boldsymbol{\phi}) = \frac{1}{N}\sum_n \log q_{\boldsymbol{\phi}}(\mathbf{x}_n). \tag{2.13}$$

From the strong law of large numbers, as $N \to \infty$ we have that $L(\boldsymbol{\phi})$ converges almost surely to $\mathbb{E}_{p(\mathbf{x})}(\log q_{\boldsymbol{\phi}}(\mathbf{x}))$. Hence, for a large enough training set, maximizing $L(\boldsymbol{\phi})$ is equivalent to minimizing $D_{\mathrm{KL}}(p(\mathbf{x}) \,\|\, q_{\boldsymbol{\phi}}(\mathbf{x}))$, since:

$$D_{\mathrm{KL}}(p(\mathbf{x}) \,\|\, q_{\boldsymbol{\phi}}(\mathbf{x})) = -\mathbb{E}_{p(\mathbf{x})}(\log q_{\boldsymbol{\phi}}(\mathbf{x})) + \text{const.} \tag{2.14}$$

Some of the merits of maximum-likelihood estimation and of KL-divergence minimization have already been discussed in section 2.1.1.

For certain simple models, the optimizer of $L(\boldsymbol{\phi})$ has a closed-form solution. For example, the maximum-likelihood parameters of a Gaussian model are the empirical mean and covariance of the training data:

$$\boldsymbol{\mu}^* = \frac{1}{N}\sum_n \mathbf{x}_n \quad \text{and} \quad \boldsymbol{\Sigma}^* = \frac{1}{N}\sum_n (\mathbf{x}_n - \boldsymbol{\mu}^*)(\mathbf{x}_n - \boldsymbol{\mu}^*)^T. \tag{2.15}$$

For mixture models based on simple parametric families, such as Gaussian mixture models, $L(\boldsymbol{\phi})$ can be (locally) maximized using the *expectation-maximization algorithm* (Dempster et al., 1977) or its online variant (Cappé and Moulines, 2008). More generally, if $L(\boldsymbol{\phi})$ is differentiable with respect to $\boldsymbol{\phi}$, it can be (locally) maximized with gradient-based methods, as I will discuss in section 2.2.4.

2.2.2 Histograms

The histogram is one of the simplest and most widely used methods for density estimation. The idea of the histogram is to partition the data space into a set of non-overlapping bins $\{B_1, \ldots, B_K\}$, and estimate $\Pr(\mathbf{x}' \in B_k)$ by the fraction of training datapoints in B_k. Then, the density in B_k is approximated by the estimate of $\Pr(\mathbf{x}' \in B_k)$ divided by the volume of B_k.

Histograms are often described as non-parametric models (e.g. Bishop, 2006, section 2.5). However, given the definition of a parametric model I gave earlier, I would argue that histograms are better described as parametric models that are trained with maximum likelihood. Given a partition of the data space into K non-overlapping bins, a histogram is the following parametric model:

$$q_{\boldsymbol{\phi}}(\mathbf{x}) = \prod_k \pi_k^{I(\mathbf{x}\in B_k)} \quad \text{where} \quad \sum_k \pi_k \left|B_k\right| = 1 \quad \text{and} \quad \pi_k \geq 0 \;\; \text{for all } k. \tag{2.16}$$

In the above, $I(\cdot)$ is the indicator function, which takes a logical statement and outputs 1 if the statement is true and 0 otherwise. Each π_k represents the density in bin B_k. The parameters of the histogram are $\boldsymbol{\phi} = \{\pi_1, \ldots, \pi_K\}$.

The average log likelihood of the histogram on training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ is:

$$L(\boldsymbol{\phi}) = \frac{1}{N}\sum_k N_k \log \pi_k, \tag{2.17}$$

where $N_k = \sum_n I(\mathbf{x}_n \in B_k)$ is the number of training datapoints in B_k. Taking into account the equality constraint $\sum_k \pi_k \left|B_k\right| = 1$, we can write the Lagrangian of the maximization problem as:

$$\mathcal{L}(\boldsymbol{\phi}, \lambda) = L(\boldsymbol{\phi}) - \lambda\left(\sum_k \pi_k \left|B_k\right| - 1\right), \tag{2.18}$$

where λ is a Lagrange multiplier enforcing the equality constraint. Taking derivatives of the Lagrangian with respect to π_k and λ and jointly solving for zero, we find that the maximum-likelihood optimizer is:

$$\pi_k^* = \frac{N_k}{N\left|B_k\right|}, \tag{2.19}$$

which also satisfies $\pi_k \geq 0$. As expected, the maximum-likelihood density in B_k is the fraction of the training data in B_k divided by the volume of B_k.

In practice, to construct the bins we typically grid up each axis between two extremes, and take the D-dimensional hyperrectangles formed this way to be the bins. How fine or coarse we grid up the space determines the volume of the bins and the granularity of the histogram. There is a

bias-variance tradeoff controlled by bin volume: a histogram with too many bins of small volume may overfit, whereas a histogram with too few bins may underfit.

A drawback of histograms is that they suffer from the curse of dimensionality. To illustrate why, suppose we are trying to estimate a uniform density in the D-dimensional unit cube $[0,1]^D$, and that we'd like a granularity of K equally sized bins per axis. The total number of bins will be K^D, which is also the expected number of datapoints until one of them falls in a given bin. Hence, the amount of training data we'd need to populate the histogram scales exponentially with dimensionality. In practice, histograms are often used in low dimensions if there are enough datapoints (e.g. for visualization purposes) but rarely in more than two or three dimensions.

2.2.3 Kernel density estimation

Kernel density estimation is a non-parametric method for estimating densities. A kernel density estimator can be thought of as a smoothed version of the empirical distribution of the training data. Given training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, their *empirical distribution* $q_0(\mathbf{x})$ is an equally weighted mixture of N delta distributions located at training datapoints:

$$q_0(\mathbf{x}) = \frac{1}{N} \sum_n \delta(\mathbf{x} - \mathbf{x}_n). \tag{2.20}$$

We can smooth out the empirical distribution and turn it into a density by replacing each delta distribution with a *smoothing kernel*. A smoothing kernel $k_\epsilon(\mathbf{u})$ is a density function defined by:

$$k_\epsilon(\mathbf{u}) = \frac{1}{\epsilon^D} k_1\Big(\frac{\mathbf{u}}{\epsilon}\Big), \tag{2.21}$$

where $\epsilon > 0$, and $k_1(\mathbf{u})$ is a density function bounded from above. The parameter ϵ controls the "width" of the kernel; as $\epsilon \to 0$, $k_\epsilon(\mathbf{u})$ approaches $\delta(\mathbf{u})$. Given a smoothing kernel $k_\epsilon(\mathbf{u})$, the *kernel density estimator* is defined as:

$$q_\epsilon(\mathbf{x}) = \frac{1}{N} \sum_n k_\epsilon(\mathbf{x} - \mathbf{x}_n). \tag{2.22}$$

In practice, common choices of kernel include the Gaussian kernel:

$$k_1(\mathbf{u}) = \frac{1}{(2\pi)^{D/2}} \exp\Big(-\frac{1}{2} \|\mathbf{u}\|^2\Big), \tag{2.23}$$

or the multiplicative Epanechnikov kernel:

$$k_1(\mathbf{u}) = \begin{cases} \left(\frac{3}{4}\right)^D \prod_d (1 - u_d^2) & |u_d| \leq 1 \text{ for all } d \\ 0 & \text{otherwise.} \end{cases} \tag{2.24}$$

The multiplicative Epanechnikov kernel is the most efficient among decomposable kernels, in the sense that asymptotically it achieves the lowest mean squared error (Epanechnikov, 1969).

In the limit $\epsilon \to 0$, the kernel density estimator is *unbiased*: it is equal to the true density in expectation. This is because as $\epsilon \to 0$ we have $q_\epsilon(\mathbf{x}) \to q_0(\mathbf{x})$, and

$$\mathbb{E}(q_0(\mathbf{x})) = \frac{1}{N} \sum_n \mathbb{E}_{p(\mathbf{x}_n)}(\delta(\mathbf{x} - \mathbf{x}_n)) = \mathbb{E}_{p(\mathbf{x}')}(\delta(\mathbf{x} - \mathbf{x}')) = p(\mathbf{x}). \tag{2.25}$$

Moreover, the kernel density estimator is *consistent*: it approaches the true density for small ϵ and large N, provided ϵ doesn't shrink too fast with N. To show this, we first upper-bound the

variance of the estimator:

$$\mathbb{V}(q_\epsilon(\mathbf{x})) = \frac{1}{N^2}\sum_n \mathbb{V}_{p(\mathbf{x}_n)}(k_\epsilon(\mathbf{x}-\mathbf{x}_n)) = \frac{1}{N}\mathbb{V}_{p(\mathbf{x}')}(k_\epsilon(\mathbf{x}-\mathbf{x}')) \tag{2.26}$$

$$\leq \frac{1}{N}\mathbb{E}_{p(\mathbf{x}')}\big(k_\epsilon^2(\mathbf{x}-\mathbf{x}')\big) = \frac{1}{N\epsilon^{2D}}\int_{\mathbb{R}^D} k_1^2\left(\frac{\mathbf{x}-\mathbf{x}'}{\epsilon}\right)p(\mathbf{x}')\,\mathrm{d}\mathbf{x}' \tag{2.27}$$

$$\leq \frac{\sup_{\mathbf{u}} k_1^2(\mathbf{u})}{N\epsilon^{2D}}. \tag{2.28}$$

We can see that the variance approaches zero as $N\epsilon^{2D}$ approaches infinity. Hence, $q_\epsilon(\mathbf{x})$ converges in probability to $p(\mathbf{x})$ as N approaches infinity, provided that ϵ approaches zero at a rate less than $N^{-1/2D}$.

In practice, the width parameter ϵ controls the degree of smoothness, and trades off bias for variance: if ϵ is too low the model may overfit, whereas if ϵ is too high the model may underfit. In general, we want ϵ to be smaller the more data we have and larger the higher the dimension is; there are rules of thumb for setting ϵ based on N and D such as *Scott's rule* (Scott, 1992) or *Silverman's rule* (Silverman, 1986).

Sometimes it is not possible to find a value for ϵ that works equally well everywhere. For instance, a lower value may be more appropriate in regions with high concentration of training data than in regions with low concentration. One possible solution, known as the *method of nearest neighbours*, is to choose a different ϵ for each location $\mathbf{x}$, such that the effective number of training datapoints contributing to the density at $\mathbf{x}$ is constant. However, the method of nearest neighbours doesn't always result in a normalizable density (Bishop, 2006, section 2.5.2).

The kernel density estimator is widely used and a strong baseline in low dimensions due to its flexibility and good asymptotic properties. However it suffers from the curse of dimensionality in high dimensions. To illustrate why, consider estimating the uniform density in the unit cube $[0,1]^D$ using the multiplicative Epanechnikov kernel, whose support is a D-dimensional hyperrectangle of side 2ϵ. The volume of space covered by kernels is at most $N(2\epsilon)^D$, which approaches zero as D grows large for any $\epsilon < 1/2$. Hence, to avoid covering only a vanishing amount of space, we must either make the support of the kernel at least as large as the support of the entire density, or have the number of training datapoints grow at least exponentially with dimensionality.

Compared to parametric methods, kernel density estimation and non-parametric methods in general have the advantage that they don't require training: there is no need to search for a model because the training data *is* the model. However, the memory cost of storing the model and the computational cost of evaluating the model grow linearly with N, which can be significant for large datasets. In contrast, parametric models have fixed memory and evaluation costs.

2.2.4 Neural density estimation

Neural density estimation is a parametric method for density estimation that uses neural networks to parameterize a density model. A *neural density estimator* is a neural network with parameters $\boldsymbol{\phi}$ that takes as input a datapoint $\mathbf{x}$ and returns a real number $f_{\boldsymbol{\phi}}(\mathbf{x})$ such that:

$$\int_{\mathbb{R}^D} \exp(f_{\boldsymbol{\phi}}(\mathbf{x}))\,\mathrm{d}\mathbf{x} = 1. \tag{2.29}$$

The above constraint is enforced by construction, that is, the architecture of the neural network is such that $\exp(f_{\boldsymbol{\phi}}(\mathbf{x}))$ integrates to 1 for all settings of $\boldsymbol{\phi}$ (I will discuss how this can be achieved in section 2.3). Since $\exp(f_{\boldsymbol{\phi}}(\mathbf{x}))$ meets all the requirements of a density function, the neural network can be used as density model $q_{\boldsymbol{\phi}}(\mathbf{x}) = \exp(f_{\boldsymbol{\phi}}(\mathbf{x}))$.

Given training data $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, neural density estimators are typically trained by maximizing the average log likelihood:

$$L(\boldsymbol{\phi}) = \frac{1}{N} \sum_n \log q_{\boldsymbol{\phi}}(\mathbf{x}_n) = \frac{1}{N} \sum_n f_{\boldsymbol{\phi}}(\mathbf{x}_n). \tag{2.30}$$

The maximization of $L(\boldsymbol{\phi})$ is typically done using a variant of *stochastic-gradient ascent* (Bottou, 2012). First, the parameters $\boldsymbol{\phi}$ are initialized to some arbitrary value. The algorithm proceeds in a number of iterations, in each of which $\boldsymbol{\phi}$ is updated. In each iteration, a subset of M training datapoints $\{\mathbf{x}_{n_1}, \ldots, \mathbf{x}_{n_M}\}$, known as a *minibatch*, is selected at random. The selection is usually done without replacement; if no more datapoints are left, all datapoints are put back in. Then, the gradient with respect to $\boldsymbol{\phi}$ of the average log likelihood on the minibatch is computed:

$$\nabla_{\boldsymbol{\phi}} \hat{L}(\boldsymbol{\phi}) = \frac{1}{M} \sum_m \nabla_{\boldsymbol{\phi}} f_{\boldsymbol{\phi}}(\mathbf{x}_{n_m}). \tag{2.31}$$

Each $\nabla_{\boldsymbol{\phi}} f_{\boldsymbol{\phi}}(\mathbf{x}_{n_m})$ can be computed in parallel using *reverse-mode automatic differentiation*, also known as *backpropagation* in the context of neural networks (Goodfellow et al., 2016, section 6.5). The gradient $\nabla_{\boldsymbol{\phi}} \hat{L}(\boldsymbol{\phi})$ is an unbiased estimator of $\nabla_{\boldsymbol{\phi}} L(\boldsymbol{\phi})$, and is known as a *stochastic gradient*. Finally, an *ascent direction* $\mathbf{d}$ is computed based on its previous value, the total number of iterations so far, the current stochastic gradient, and possibly a window (or running aggregate) of previous stochastic gradients, and the parameters are updated by $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \mathbf{d}$. There are various strategies for computing $\mathbf{d}$, such as *momentum* (Qian, 1999), *AdaGrad* (Duchi et al., 2011), *AdaDelta* (Zeiler, 2012), *Adam* (Kingma and Ba, 2015), and *AMSGrad* (Reddi et al., 2018).

Stochastic-gradient ascent is a general algorithm for optimizing differentiable functions that can be written as averages of multiple terms. Due to its generality, it has the advantage that it decouples the task of modelling the density from the task of optimizing the training objective. Due to its use of stochastic gradients instead of full gradients, it scales well to large datasets, and it can be used with training datasets of infinite size (such as data produced by a generative process on the fly). Finally, there is some preliminary evidence that the stochasticity of the gradients may contribute in finding parameter settings that generalize well (Keskar et al., 2017).

The question that remains to be answered is how we can design neural networks such that their exponentiated output integrates to 1 by construction. This is one of the main contributions of this thesis, and it will be the topic of section 2.3. As we shall see, it is possible to design neural density estimators that, although parametric, are flexible enough to approximate complex densities in thousands of dimensions.

2.3 The paper

This section presents the paper *Masked Autoregressive Flow for Density Estimation*, which is the main contribution of this chapter. The paper discusses state-of-the-art methods for constructing neural density estimators, and proposes a new method which we term *Masked Autoregressive Flow*. We show how Masked Autoregressive Flow can increase the flexibility of previously proposed neural density estimators, and demonstrate MAF's performance in high-dimensional density estimation.

The paper was initially published as a preprint on arXiv in May 2017. Then, it was accepted for publication at the conference *Advances in Neural Information Processing Systems (NeurIPS)* in December 2017. It was featured as an oral presentation at the conference; of the 3,240 papers submitted to NeurIPS in 2017, 678 were accepted for publication, of which 40 were featured as oral presentations.

Author contributions

The paper is co-authored by me, Theo Pavlakou and Iain Murray. As the leading author, I conceived and developed Masked Autoregressive Flow, performed the experiments, and wrote the paper. Theo Pavlakou prepared the UCI datasets used in section 4.2 of the paper; his earlier work on density estimation using the UCI datasets served as a guide and point of reference for the experiments in the paper. Iain Murray supervised the project, offered suggestions, and helped revise the final version.

Differences in notation

The previous sections used $p(\mathbf{x})$ to mean the true density of the generative process and $q_{\boldsymbol{\phi}}(\mathbf{x})$ to mean the density represented by a parametric model with parameters $\boldsymbol{\phi}$. The paper uses $p(\mathbf{x})$ both for the true density and for the model density depending on context. In section 3.2 and appendix A of the paper, where disambiguation between the two densities is required, we use $\pi_x(\mathbf{x})$ for the true density and $p_x(\mathbf{x})$ for the model density.

Corrections from original version

About a year after the initial publication of the paper, we discovered that the experimental results on conditional density estimation with Masked Autoregressive Flow were incorrect due to an error in the code. Upon discovery, we corrected the results and issued a replacement of the paper on arXiv. The version included in this chapter is the corrected version of the paper, which was published on arXiv in June 2018. The particular results that were updated from earlier versions are indicated with a footnote in this version.

Masked Autoregressive Flow for Density Estimation

George Papamakarios
University of Edinburgh
g.papamakarios@ed.ac.uk

Theo Pavlakou
University of Edinburgh
theo.pavlakou@ed.ac.uk

Iain Murray
University of Edinburgh
i.murray@ed.ac.uk

Abstract

Autoregressive models are among the best performing neural density estimators. We describe an approach for increasing the flexibility of an autoregressive model, based on modelling the random numbers that the model uses internally when generating data. By constructing a stack of autoregressive models, each modelling the random numbers of the next model in the stack, we obtain a type of normalizing flow suitable for density estimation, which we call Masked Autoregressive Flow. This type of flow is closely related to Inverse Autoregressive Flow and is a generalization of Real NVP. Masked Autoregressive Flow achieves state-of-the-art performance in a range of general-purpose density estimation tasks.

1 Introduction

The joint density $p(\mathbf{x})$ of a set of variables $\mathbf{x}$ is a central object of interest in machine learning. Being able to access and manipulate $p(\mathbf{x})$ enables a wide range of tasks to be performed, such as inference, prediction, data completion and data generation. As such, the problem of estimating $p(\mathbf{x})$ from a set of examples $\{\mathbf{x}_n\}$ is at the core of probabilistic unsupervised learning and generative modelling.

In recent years, using neural networks for density estimation has been particularly successful. Combining the flexibility and learning capacity of neural networks with prior knowledge about the structure of data to be modelled has led to impressive results in modelling natural images [6, 31, 34, 41, 42] and audio data [38, 40]. State-of-the-art neural density estimators have also been used for likelihood-free inference from simulated data [24, 26], variational inference [16, 27], and as surrogates for maximum entropy models [22].

Neural density estimators differ from other approaches to generative modelling—such as variational autoencoders [15, 28] and generative adversarial networks [10]—in that they readily provide exact density evaluations. As such, they are more suitable in applications where the focus is on explicitly evaluating densities, rather than generating synthetic data. For instance, density estimators can learn suitable priors for data from large unlabelled datasets, for use in standard Bayesian inference [43]. In simulation-based likelihood-free inference, conditional density estimators can learn models for the likelihood [7] or the posterior [26] from simulated data. Density estimators can learn effective proposals for importance sampling [25] or sequential Monte Carlo [11, 24]; such proposals can be used in probabilistic programming environments to speed up inference [18, 19]. Finally, conditional density estimators can be used as flexible inference networks for amortized variational inference and as part of variational autoencoders [15, 28].

A challenge in neural density estimation is to construct models that are flexible enough to represent complex densities, but have tractable density functions and learning algorithms. There are mainly two families of neural density estimators that are both flexible and tractable: *autoregressive models* [39] and *normalizing flows* [27]. Autoregressive models decompose the joint density as a product of conditionals, and model each conditional in turn. Normalizing flows transform a base density (e.g. a standard Gaussian) into the target density by an invertible transformation with tractable Jacobian.

Our starting point is the realization (as pointed out by Kingma et al. [16]) that autoregressive models, when used to generate data, correspond to a differentiable transformation of an external source of

randomness (typically obtained by random number generators). This transformation has a tractable Jacobian by design, and for certain autoregressive models it is also invertible, hence it precisely corresponds to a normalizing flow. Viewing an autoregressive model as a normalizing flow opens the possibility of increasing its flexibility by stacking multiple models of the same type, by having each model provide the source of randomness for the next model in the stack. The resulting stack of models is a normalizing flow that is more flexible than the original model, and that remains tractable.

In this paper we present *Masked Autoregressive Flow* (MAF), which is a particular implementation of the above normalizing flow that uses the Masked Autoencoder for Distribution Estimation (MADE) [9] as a building block. The use of MADE enables density evaluations without the sequential loop that is typical of autoregressive models, and thus makes MAF fast to evaluate and train on parallel computing architectures such as Graphics Processing Units (GPUs). We show a close theoretical connection between MAF and Inverse Autoregressive Flow (IAF) [16], which has been designed for variational inference instead of density estimation, and show that both correspond to generalizations of the successful Real NVP [6]. We experimentally evaluate MAF on a wide range of datasets, and we demonstrate that MAF outperforms RealNVP and achieves state-of-the-art performance on a variety of general-purpose density estimation tasks.

2 Background

2.1 Autoregressive density estimation

Using the chain rule of probability, any joint density $p(\mathbf{x})$ can be decomposed into a product of one-dimensional conditionals as $p(\mathbf{x}) = \prod_i p(x_i \mid \mathbf{x}_{1:i-1})$. Autoregressive density estimators [39] model each conditional $p(x_i \mid \mathbf{x}_{1:i-1})$ as a parametric density, whose parameters are a function of a hidden state $\mathbf{h}_i$. In recurrent architectures, $\mathbf{h}_i$ is a function of the previous hidden state $\mathbf{h}_{i-1}$ and the i^{th} input variable x_i. The Real-valued Neural Autoregressive Density Estimator (RNADE) [36] uses mixtures of Gaussian or Laplace densities for modelling the conditionals, and a simple linear rule for updating the hidden state. More flexible approaches for updating the hidden state are based on Long Short-Term Memory recurrent neural networks [34, 42].

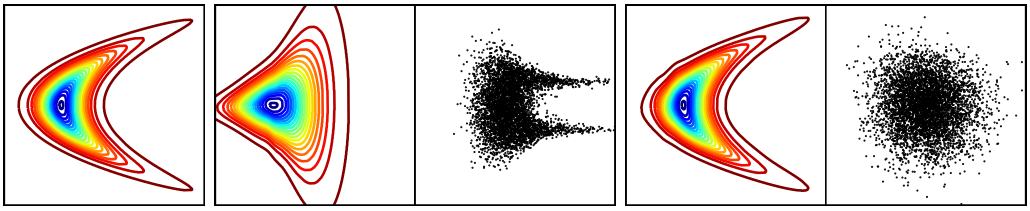
A drawback of autoregressive models is that they are sensitive to the order of the variables. For example, the order of the variables matters when learning the density of Figure 1a if we assume a model with Gaussian conditionals. As Figure 1b shows, a model with order (x_1, x_2) cannot learn this density, even though the same model with order (x_2, x_1) can represent it perfectly. In practice is it hard to know which of the factorially many orders is the most suitable for the task at hand. Autoregressive models that are trained to work with an order chosen at random have been developed, and the predictions from different orders can then be combined in an ensemble [9, 37]. Our approach (Section 3) can use a different order in each layer, and using random orders would also be possible.

Straightforward recurrent autoregressive models would update a hidden state sequentially for every variable, requiring D sequential computations to compute the probability $p(\mathbf{x})$ of a D-dimensional vector, which is not well-suited for computation on parallel architectures such as GPUs. One way to enable parallel computation is to start with a fully-connected model with D inputs and D outputs, and drop out connections in order to ensure that output i will only be connected to inputs $1, 2, \ldots, i-1$. Output i can then be interpreted as computing the parameters of the i^{th} conditional $p(x_i \mid \mathbf{x}_{1:i-1})$. By construction, the resulting model will satisfy the autoregressive property, and at the same time it will be able to calculate $p(\mathbf{x})$ efficiently on a GPU. An example of this approach is the Masked Autoencoder for Distribution Estimation (MADE) [9], which drops out connections by multiplying the weight matrices of a fully-connected autoencoder with binary masks. Other mechanisms for dropping out connections include masked convolutions [42] and causal convolutions [40].

2.2 Normalizing flows

A normalizing flow [27] represents $p(\mathbf{x})$ as an invertible differentiable transformation f of a base density $\pi_u(\mathbf{u})$. That is, $\mathbf{x} = f(\mathbf{u})$ where $\mathbf{u} \sim \pi_u(\mathbf{u})$. The base density $\pi_u(\mathbf{u})$ is chosen such that it can be easily evaluated for any input $\mathbf{u}$ (a common choice for $\pi_u(\mathbf{u})$ is a standard Gaussian). Under the invertibility assumption for f, the density $p(\mathbf{x})$ can be calculated as

$$p(\mathbf{x}) = \pi_u\big(f^{-1}(\mathbf{x})\big) \left| \det \left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right|. \tag{1}$$



(a) Target density (b) MADE with Gaussian conditionals (c) MAF with 5 layers

Figure 1: **(a)** The density to be learnt, defined as $p(x_1, x_2) = \mathcal{N}(x_2 \,|\, 0, 4)\mathcal{N}\big(x_1 \,|\, \frac{1}{4}x_2^2, 1\big)$. **(b)** The density learnt by a MADE with order (x_1, x_2) and Gaussian conditionals. Scatter plot shows the train data transformed into random numbers $\mathbf{u}$; the non-Gaussian distribution indicates that the model is a poor fit. **(c)** Learnt density and transformed train data of a 5 layer MAF with the same order (x_1, x_2).

In order for Equation (1) to be tractable, the transformation f must be constructed such that (a) it is easy to invert, and (b) the determinant of its Jacobian is easy to compute. An important point is that if transformations f_1 and f_2 have the above properties, then their composition $f_1 \circ f_2$ also has these properties. In other words, the transformation f can be made deeper by composing multiple instances of it, and the result will still be a valid normalizing flow.

There have been various approaches in developing normalizing flows. An early example is Gaussianization [4], which is based on successive application of independent component analysis. Enforcing invertibility with nonsingular weight matrices has been proposed [3, 29], however in such approaches calculating the determinant of the Jacobian scales cubicly with data dimensionality in general. Planar/radial flows [27] and Inverse Autoregressive Flow (IAF) [16] are models whose Jacobian is tractable by design. However, they were developed primarily for variational inference and are not well-suited for density estimation, as they can only efficiently calculate the density of their own samples and not of externally provided datapoints. The Non-linear Independent Components Estimator (NICE) [5] and its successor Real NVP [6] have a tractable Jacobian and are also suitable for density estimation. IAF, NICE and Real NVP are discussed in more detail in Section 3.

3 Masked Autoregressive Flow

3.1 Autoregressive models as normalizing flows

Consider an autoregressive model whose conditionals are parameterized as single Gaussians. That is, the i^{th} conditional is given by

$$p(x_i \,|\, \mathbf{x}_{1:i-1}) = \mathcal{N}\big(x_i \,|\, \mu_i, (\exp \alpha_i)^2\big) \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{x}_{1:i-1}) \;\text{ and }\; \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}). \quad (2)$$

In the above, f_{μ_i} and f_{α_i} are unconstrained scalar functions that compute the mean and log standard deviation of the i^{th} conditional given all previous variables. We can generate data from the above model using the following recursion:

$$x_i = u_i \exp \alpha_i + \mu_i \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{x}_{1:i-1}),\;\; \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}) \;\text{ and }\; u_i \sim \mathcal{N}(0, 1). \quad (3)$$

In the above, $\mathbf{u} = (u_1, u_2, \ldots, u_I)$ is the vector of random numbers the model uses internally to generate data, typically by making calls to a random number generator often called `randn()`.

Equation (3) provides an alternative characterization of the autoregressive model as a transformation f from the space of random numbers $\mathbf{u}$ to the space of data $\mathbf{x}$. That is, we can express the model as $\mathbf{x} = f(\mathbf{u})$ where $\mathbf{u} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. By construction, f is easily invertible. Given a datapoint $\mathbf{x}$, the random numbers $\mathbf{u}$ that were used to generate it are obtained by the following recursion:

$$u_i = (x_i - \mu_i) \exp(-\alpha_i) \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{x}_{1:i-1}) \;\text{ and }\; \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}). \quad (4)$$

Due to the autoregressive structure, the Jacobian of f^{-1} is triangular by design, hence its absolute determinant can be easily obtained as follows:

$$\left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right| = \exp\Big(-\sum\nolimits_i \alpha_i\Big) \quad \text{where} \quad \alpha_i = f_{\alpha_i}(\mathbf{x}_{1:i-1}). \quad (5)$$

It follows that the autoregressive model can be equivalently interpreted as a normalizing flow, whose density $p(\mathbf{x})$ can be obtained by substituting Equations (4) and (5) into Equation (1). This observation was first pointed out by Kingma et al. [16].

A useful diagnostic for assessing whether an autoregressive model of the above type fits the target density well is to transform the train data $\{\mathbf{x}_n\}$ into corresponding random numbers $\{\mathbf{u}_n\}$ using Equation (4), and assess whether the u_i's come from independent standard normals. If the u_i's do not seem to come from independent standard normals, this is evidence that the model is a bad fit. For instance, Figure 1b shows that the scatter plot of the random numbers associated with the train data can look significantly non-Gaussian if the model fits the target density poorly.

Here we interpret autoregressive models as a flow, and improve the model fit by stacking multiple instances of the model into a deeper flow. Given autoregressive models $M_1, M_2, \ldots, M_K$, we model the density of the random numbers $\mathbf{u}_1$ of M_1 with M_2, model the random numbers $\mathbf{u}_2$ of M_2 with M_3 and so on, finally modelling the random numbers $\mathbf{u}_K$ of M_K with a standard Gaussian. This stacking adds flexibility: for example, Figure 1c demonstrates that a flow of 5 autoregressive models is able to learn multimodal conditionals, even though each model has unimodal conditionals. Stacking has previously been used in a similar way to improve model fit of deep belief nets [12] and deep mixtures of factor analyzers [32].

We choose to implement the set of functions $\{f_{\mu_i}, f_{\alpha_i}\}$ with *masking*, following the approach used by MADE [9]. MADE is a feedforward network that takes $\mathbf{x}$ as input and outputs μ_i and α_i for all i with a single forward pass. The autoregressive property is enforced by multiplying the weight matrices of MADE with suitably constructed binary masks. In other words, we use MADE with Gaussian conditionals as the building layer of our flow. The benefit of using masking is that it enables transforming from data $\mathbf{x}$ to random numbers $\mathbf{u}$ and thus calculating $p(\mathbf{x})$ in one forward pass through the flow, thus eliminating the need for sequential recursion as in Equation (4). We call this implementation of stacking MADEs into a flow *Masked Autoregressive Flow* (MAF).

3.2 Relationship with Inverse Autoregressive Flow

Like MAF, Inverse Autoregressive Flow (IAF) [16] is a normalizing flow which uses MADE as its component layer. Each layer of IAF is defined by the following recursion:

$$x_i = u_i \exp \alpha_i + \mu_i \quad \text{where} \quad \mu_i = f_{\mu_i}(\mathbf{u}_{1:i-1}) \;\text{ and }\; \alpha_i = f_{\alpha_i}(\mathbf{u}_{1:i-1}). \tag{6}$$

Similarly to MAF, functions $\{f_{\mu_i}, f_{\alpha_i}\}$ are computed using a MADE with Gaussian conditionals. The difference is architectural: in MAF μ_i and α_i are directly computed from previous *data variables* $\mathbf{x}_{1:i-1}$, whereas in IAF μ_i and α_i are directly computed from previous *random numbers* $\mathbf{u}_{1:i-1}$.

The consequence of the above is that MAF and IAF are different models with different computational trade-offs. MAF is capable of calculating the density $p(\mathbf{x})$ of *any* datapoint $\mathbf{x}$ in one pass through the model, however sampling from it requires performing D sequential passes (where D is the dimensionality of $\mathbf{x}$). In contrast, IAF can generate samples and calculate their density with one pass, however calculating the density $p(\mathbf{x})$ of an externally provided datapoint $\mathbf{x}$ requires D passes to find the random numbers $\mathbf{u}$ associated with $\mathbf{x}$. Hence, the design choice of whether to connect μ_i and α_i directly to $\mathbf{x}_{1:i-1}$ (obtaining MAF) or to $\mathbf{u}_{1:i-1}$ (obtaining IAF) depends on the intended usage. IAF is suitable as a recognition model for stochastic variational inference [15, 28], where it only ever needs to calculate the density of its own samples. In contrast, MAF is more suitable for density estimation, because each example requires only one pass through the model whereas IAF requires D.

A theoretical equivalence between MAF and IAF is that training a MAF with maximum likelihood corresponds to fitting an implicit IAF to the base density with stochastic variational inference. Let $\pi_x(\mathbf{x})$ be the data density we wish to learn, $\pi_u(\mathbf{u})$ be the base density, and f be the transformation from $\mathbf{u}$ to $\mathbf{x}$ as implemented by MAF. The density defined by MAF (with added subscript x for disambiguation) is

$$p_x(\mathbf{x}) = \pi_u\big(f^{-1}(\mathbf{x})\big) \left| \det \left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right|. \tag{7}$$

The inverse transformation f^{-1} from $\mathbf{x}$ to $\mathbf{u}$ can be seen as describing an implicit IAF with base density $\pi_x(\mathbf{x})$, which defines the following implicit density over the $\mathbf{u}$ space:

$$p_u(\mathbf{u}) = \pi_x(f(\mathbf{u})) \left| \det \left(\frac{\partial f}{\partial \mathbf{u}} \right) \right|. \tag{8}$$

Training MAF by maximizing the total log likelihood $\sum_n \log p(\mathbf{x}_n)$ on train data $\{\mathbf{x}_n\}$ corresponds to fitting $p_x(\mathbf{x})$ to $\pi_x(\mathbf{x})$ by stochastically minimizing $D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x}))$. In Section A of the appendix, we show that

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u})). \tag{9}$$

Hence, stochastically minimizing $D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x}))$ is equivalent to fitting $p_u(\mathbf{u})$ to $\pi_u(\mathbf{u})$ by minimizing $D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u}))$. Since the latter is the loss function used in variational inference, and $p_u(\mathbf{u})$ can be seen as an IAF with base density $\pi_x(\mathbf{x})$ and transformation f^{-1}, it follows that training MAF as a density estimator of $\pi_x(\mathbf{x})$ is equivalent to performing stochastic variational inference with an implicit IAF, where the posterior is taken to be the base density $\pi_u(\mathbf{u})$ and the transformation f^{-1} implements the reparameterization trick [15, 28]. This argument is presented in more detail in Section A of the appendix.

3.3 Relationship with Real NVP

Real NVP [6] (NVP stands for Non Volume Preserving) is a normalizing flow obtained by stacking *coupling layers*. A coupling layer is an invertible transformation f from random numbers $\mathbf{u}$ to data $\mathbf{x}$ with a tractable Jacobian, defined by

$$\begin{aligned} \mathbf{x}_{1:d} &= \mathbf{u}_{1:d} \\ \mathbf{x}_{d+1:D} &= \mathbf{u}_{d+1:D} \odot \exp \boldsymbol{\alpha} + \boldsymbol{\mu} \end{aligned} \qquad \text{where} \qquad \begin{aligned} \boldsymbol{\mu} &= f_\mu(\mathbf{u}_{1:d}) \\ \boldsymbol{\alpha} &= f_\alpha(\mathbf{u}_{1:d}). \end{aligned} \tag{10}$$

In the above, $\odot$ denotes elementwise multiplication, and the $\exp$ is applied to each element of $\boldsymbol{\alpha}$. The transformation copies the first d elements, and scales and shifts the remaining $D-d$ elements, with the amount of scaling and shifting being a function of the first d elements. When stacking coupling layers into a flow, the elements are permuted across layers so that a different set of elements is copied each time. A special case of the coupling layer where $\boldsymbol{\alpha}=\mathbf{0}$ is used by NICE [5].

We can see that the coupling layer is a special case of both the autoregressive transformation used by MAF in Equation (3), and the autoregressive transformation used by IAF in Equation (6). Indeed, we can recover the coupling layer from the autoregressive transformation of MAF by setting $\mu_i = \alpha_i = 0$ for $i \leq d$ and making μ_i and α_i functions of only $\mathbf{x}_{1:d}$ for $i > d$ (for IAF we need to make μ_i and α_i functions of $\mathbf{u}_{1:d}$ instead for $i > d$). In other words, both MAF and IAF can be seen as more flexible (but different) generalizations of Real NVP, where each element is individually scaled and shifted as a function of all previous elements. The advantage of Real NVP compared to MAF and IAF is that it can both generate data and estimate densities with one forward pass only, whereas MAF would need D passes to generate data and IAF would need D passes to estimate densities.

3.4 Conditional MAF

Given a set of example pairs $\{(\mathbf{x}_n, \mathbf{y}_n)\}$, conditional density estimation is the task of estimating the conditional density $p(\mathbf{x} \,|\, \mathbf{y})$. Autoregressive modelling extends naturally to conditional density estimation. Each term in the chain rule of probability can be conditioned on side-information $\mathbf{y}$, decomposing any conditional density as $p(\mathbf{x} \,|\, \mathbf{y}) = \prod_i p(x_i \,|\, \mathbf{x}_{1:i-1}, \mathbf{y})$. Therefore, we can turn any unconditional autoregressive model into a conditional one by augmenting its set of input variables with $\mathbf{y}$ and only modelling the conditionals that correspond to $\mathbf{x}$. Any order of the variables can be chosen, as long as $\mathbf{y}$ comes before $\mathbf{x}$. In masked autoregressive models, no connections need to be dropped from the $\mathbf{y}$ inputs to the rest of the network.

We can implement a conditional version of MAF by stacking MADEs that were made conditional using the above strategy. That is, in a conditional MAF, the vector $\mathbf{y}$ becomes an additional input for every layer. As a special case of MAF, Real NVP can be made conditional in the same way.

4 Experiments

4.1 Implementation and setup

We systematically evaluate three types of density estimator (MADE, Real NVP and MAF) in terms of density estimation performance on a variety of datasets. Code for reproducing our experiments (which uses Theano [33]) can be found at `https://github.com/gpapamak/maf`.

MADE. We consider two versions: (a) a MADE with Gaussian conditionals, denoted simply by MADE, and (b) a MADE whose conditionals are each parameterized as a mixture of C Gaussians, denoted by MADE MoG. We used $C = 10$ in all our experiments. MADE can be seen either as a MADE MoG with $C = 1$, or as a MAF with only one autoregressive layer. Adding more Gaussian components per conditional or stacking MADEs to form a MAF are two alternative ways of increasing the flexibility of MADE, which we are interested in comparing.

Real NVP. We consider a general-purpose implementation of the coupling layer, which uses two feedforward neural networks, implementing the scaling function f_α and the shifting function f_μ respectively. Both networks have the same architecture, except that f_α has hyperbolic tangent hidden units, whereas f_μ has rectified linear hidden units (we found this combination to perform best). Both networks have a linear output. We consider Real NVPs with either 5 or 10 coupling layers, denoted by Real NVP (5) and Real NVP (10) respectively, and in both cases the base density is a standard Gaussian. Successive coupling layers alternate between (a) copying the odd-indexed variables and transforming the even-indexed variables, and (b) copying the even-indexed variables and transforming the odd-indexed variables.

It is important to clarify that this is a general-purpose implementation of Real NVP which is different and thus not comparable to its original version [6], which was designed specifically for image data. Here we are interested in comparing coupling layers with autoregressive layers as building blocks of normalizing flows for general-purpose density estimation tasks, and our design of Real NVP is such that a fair comparison between the two can be made.

MAF. We consider three versions: (a) a MAF with 5 autoregressive layers and a standard Gaussian as a base density $\pi_u(\mathbf{u})$, denoted by MAF (5), (b) a MAF with 10 autoregressive layers and a standard Gaussian as a base density, denoted by MAF (10), and (c) a MAF with 5 autoregressive layers and a MADE MoG with $C = 10$ Gaussian components in each conditional as a base density, denoted by MAF MoG (5). MAF MoG (5) can be thought of as a MAF (5) stacked on top of a MADE MoG and trained jointly with it.

In all experiments, MADE and MADE MoG order the inputs using the order that comes with the dataset by default; no alternative orders were considered. MAF uses the default order for the first autoregressive layer (i.e. the layer that directly models the data) and reverses the order for each successive layer (the same was done for IAF by Kingma et al. [16]).

MADE, MADE MoG and each layer in MAF is a feedforward neural network with masked weight matrices, such that the autoregressive property holds. The procedure for designing the masks (due to Germain et al. [9]) is as follows. Each input or hidden unit is assigned a degree, which is an integer ranging from 1 to D, where D is the data dimensionality. The degree of an input is taken to be its index in the order. The D outputs have degrees that sequentially range from 0 to $D-1$. A unit is allowed to receive input only from units with lower or equal degree, which enforces the autoregressive property. In order for output i to be connected to all inputs with degree less than i, and thus make sure that no conditional independences are introduced, it is both necessary and sufficient that every hidden layer contains every degree. In all experiments except for CIFAR-10, we sequentially assign degrees within each hidden layer and use enough hidden units to make sure that all degrees appear. Because CIFAR-10 is high-dimensional, we used fewer hidden units than inputs and assigned degrees to hidden units uniformly at random (as was done by Germain et al. [9]).

We added batch normalization [13] after each coupling layer in Real NVP and after each autoregressive layer in MAF. Batch normalization is an elementwise scaling and shifting operation, which is easily invertible and has a tractable Jacobian, and thus it is suitable for use in a normalizing flow. We found that batch normalization in Real NVP and MAF reduces training time, increases stability during training and improves performance (a fact that was also observed by Dinh et al. [6] for Real NVP). Section B of the appendix discusses our implementation of batch normalization and its use in normalizing flows.

All models were trained with the Adam optimizer [14], using a minibatch size of 100, and a step size of 10^{-3} for MADE and MADE MoG, and of 10^{-4} for Real NVP and MAF. A small amount of ℓ_2 regularization was added, with coefficient 10^{-6}. Each model was trained with early stopping until no improvement occurred for 30 consecutive epochs on the validation set. For each model, we selected the number of hidden layers and number of hidden units based on validation performance (we gave the same options to all models), as described in Section D of the appendix.

Table 1: Average test log likelihood (in nats) for unconditional density estimation. The best performing model for each dataset is shown in bold (multiple models are highlighted if the difference is not statistically significant according to a paired t-test). Error bars correspond to 2 standard deviations.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
Gaussian	-7.74 ± 0.02	-3.58 ± 0.75	-27.93 ± 0.02	-37.24 ± 1.07	96.67 ± 0.25
MADE	-3.08 ± 0.03	3.56 ± 0.04	-20.98 ± 0.02	-15.59 ± 0.50	148.85 ± 0.28
MADE MoG	$\mathbf{0.40 \pm 0.01}$	8.47 ± 0.02	$\mathbf{-15.15 \pm 0.02}$	-12.27 ± 0.47	153.71 ± 0.28
Real NVP (5)	-0.02 ± 0.01	4.78 ± 1.80	-19.62 ± 0.02	-13.55 ± 0.49	152.97 ± 0.28
Real NVP (10)	0.17 ± 0.01	8.33 ± 0.14	-18.71 ± 0.02	-13.84 ± 0.52	153.28 ± 1.78
MAF (5)	0.14 ± 0.01	9.07 ± 0.02	-17.70 ± 0.02	$\mathbf{-11.75 \pm 0.44}$	155.69 ± 0.28
MAF (10)	0.24 ± 0.01	$\mathbf{10.08 \pm 0.02}$	-17.73 ± 0.02	-12.24 ± 0.45	154.93 ± 0.28
MAF MoG (5)	0.30 ± 0.01	9.59 ± 0.02	-17.39 ± 0.02	$\mathbf{-11.68 \pm 0.44}$	$\mathbf{156.36 \pm 0.28}$

4.2 Unconditional density estimation

Following Uria et al. [36], we perform unconditional density estimation on four UCI datasets (POWER, GAS, HEPMASS, MINIBOONE) and on a dataset of natural image patches (BSDS300).

UCI datasets. These datasets were taken from the UCI machine learning repository [21]. We selected different datasets than Uria et al. [36], because the ones they used were much smaller, resulting in an expensive cross-validation procedure involving a separate hyperparameter search for each fold. However, our data preprocessing follows Uria et al. [36]. The sample mean was subtracted from the data and each feature was divided by its sample standard deviation. Discrete-valued attributes were eliminated, as well as every attribute with a Pearson correlation coefficient greater than 0.98. These procedures are meant to avoid trivial high densities, which would make the comparison between approaches hard to interpret. Section D of the appendix gives more details about the UCI datasets and the individual preprocessing done on each of them.

Image patches. This dataset was obtained by extracting random 8×8 monochrome patches from the BSDS300 dataset of natural images [23]. We used the same preprocessing as by Uria et al. [36]. Uniform noise was added to dequantize pixel values, which was then rescaled to be in the range $[0, 1]$. The mean pixel value was subtracted from each patch, and the bottom-right pixel was discarded.

Table 1 shows the performance of each model on each dataset. A Gaussian fitted to the train data is reported as a baseline. We can see that on 3 out of 5 datasets MAF is the best performing model, with MADE MoG being the best performing model on the other 2. On all datasets, MAF outperforms Real NVP. For the MINIBOONE dataset, due to overlapping error bars, a pairwise comparison was done to determine which model performs the best, the results of which are reported in Section E of the appendix. MAF MoG (5) achieves the best reported result on BSDS300 for a single model with 156.36 nats, followed by Deep RNADE [37] with 155.2. An ensemble of 32 Deep RNADEs was reported to achieve 157.0 nats [37]. The UCI datasets were used for the first time in the literature for density estimation, so no comparison with existing work can be made yet.

4.3 Conditional density estimation

For conditional density estimation, we used the MNIST dataset of handwritten digits [20] and the CIFAR-10 dataset of natural images [17]. In both datasets, each datapoint comes from one of 10 distinct classes. We represent the class label as a 10-dimensional, one-hot encoded vector $\mathbf{y}$, and we model the density $p(\mathbf{x} \mid \mathbf{y})$, where $\mathbf{x}$ represents an image. At test time, we evaluate the probability of a test image $\mathbf{x}$ by $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x} \mid \mathbf{y}) p(\mathbf{y})$, where $p(\mathbf{y}) = \frac{1}{10}$ is a uniform prior over the labels. For comparison, we also train every model as an unconditional density estimator and report both results.

For both MNIST and CIFAR-10, we use the same preprocessing as by Dinh et al. [6]. We dequantize pixel values by adding uniform noise, and then rescale them to $[0, 1]$. We transform the rescaled pixel values into logit space by $\mathbf{x} \mapsto \text{logit}(\lambda + (1 - 2\lambda)\mathbf{x})$, where $\lambda = 10^{-6}$ for MNIST and $\lambda = 0.05$ for CIFAR-10, and perform density estimation in that space. In the case of CIFAR-10, we also augment the train set with horizontal flips of all train examples (as also done by Dinh et al. [6]).

Table 2: Average test log likelihood (in nats) for conditional density estimation. The best performing model for each dataset is shown in bold. Error bars correspond to 2 standard deviations.

	MNIST		CIFAR-10	
	unconditional	conditional	unconditional	conditional
Gaussian	-1366.9 ± 1.4	-1344.7 ± 1.8	2367 ± 29	2030 ± 41
MADE	-1380.8 ± 4.8	-1361.9 ± 1.9	147 ± 20	187 ± 20
MADE MoG	$\mathbf{-1038.5 \pm 1.8}$	$\mathbf{-1030.3 \pm 1.7}$	-397 ± 21	-119 ± 20
Real NVP (5)	-1323.2 ± 6.6	-1326.3 ± 5.8	2576 ± 27	2642 ± 26
Real NVP (10)	-1370.7 ± 10.1	-1371.3 ± 43.9	2568 ± 26	2475 ± 25
MAF (5)	-1300.5 ± 1.7	-1302.9 ± 1.7*	2936 ± 27	2983 ± 26*
MAF (10)	-1313.1 ± 2.0	-1316.8 ± 1.8*	$\mathbf{3049 \pm 26}$	$\mathbf{3058 \pm 26}$*
MAF MoG (5)	-1100.3 ± 1.6	-1092.3 ± 1.7	2911 ± 26	2936 ± 26

Table 2 shows the results on MNIST and CIFAR-10. The performance of a class-conditional Gaussian is reported as a baseline for the conditional case. Log likelihoods are calculated in logit space. MADE MoG is the best performing model on MNIST, whereas MAF is the best performing model on CIFAR-10. On CIFAR-10, both MADE and MADE MoG performed significantly worse than the Gaussian baseline. MAF outperforms Real NVP in all cases. To facilitate comparison with the literature, Section E of the appendix reports results in bits/pixel.[1]

5 Discussion

We showed that we can improve MADE by modelling the density of its internal random numbers. Alternatively, MADE can be improved by increasing the flexibility of its conditionals. The comparison between MAF and MADE MoG showed that the best approach is dataset specific; in our experiments MAF outperformed MADE MoG in 5 out of 9 cases, which is strong evidence of its competitiveness. MADE MoG is a universal density approximator; with sufficiently many hidden units and Gaussian components, it can approximate any continuous density arbitrarily well. It is an open question whether MAF with a Gaussian base density has a similar property (MAF MoG clearly does).

We also showed that the coupling layer used in Real NVP is a special case of the autoregressive layer used in MAF. In fact, MAF outperformed Real NVP in all our experiments. Real NVP has achieved impressive performance in image modelling by incorporating knowledge about image structure. Our results suggest that replacing coupling layers with autoregressive layers in the original version of Real NVP is a promising direction for further improving its performance. Real NVP maintains however the advantage over MAF (and autoregressive models in general) that samples from the model can be generated efficiently in parallel.

Density estimation is one of several types of generative modelling, with the focus on obtaining accurate densities. However, we know that accurate densities do not necessarily imply good performance in other tasks, such as in data generation [35]. Alternative approaches to generative modelling include variational autoencoders [15, 28], which are capable of efficient inference of their (potentially interpretable) latent space, and generative adversarial networks [10], which are capable of high quality data generation. Choice of method should be informed by whether the application at hand calls for accurate densities, latent space inference or high quality samples. Masked Autoregressive Flow is a contribution towards the first of these goals.

Acknowledgments

We thank Maria Gorinova for useful comments, and Johann Brehmer for discovering the error in the calculation of the test log likelihood for conditional MAF. George Papamakarios and Theo Pavlakou were supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1) and the University of Edinburgh. George Papamakarios was also supported by Microsoft Research through its PhD Scholarship Programme.

[1] In earlier versions of the paper, results marked with * were reported incorrectly, due to an error in the calculation of the test log likelihood of conditional MAF. Thus error has been corrected in the current version.

A Equivalence between MAF and IAF

In this section, we present the equivalence between MAF and IAF in full mathematical detail. Let $\pi_x(\mathbf{x})$ be the true density the train data $\{\mathbf{x}_n\}$ is sampled from. Suppose we have a MAF whose base density is $\pi_u(\mathbf{u})$, and whose transformation from $\mathbf{u}$ to $\mathbf{x}$ is f. The MAF defines the following density over the $\mathbf{x}$ space:

$$p_x(\mathbf{x}) = \pi_u\big(f^{-1}(\mathbf{x})\big) \left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right|. \tag{11}$$

Using the definition of $p_x(\mathbf{x})$ in Equation (11), we can write the Kullback–Leibler divergence from $\pi_x(\mathbf{x})$ to $p_x(\mathbf{x})$ as follows:

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = \mathbb{E}_{\pi_x(\mathbf{x})}(\log \pi_x(\mathbf{x}) - \log p_x(\mathbf{x})) \tag{12}$$

$$= \mathbb{E}_{\pi_x(\mathbf{x})}\left(\log \pi_x(\mathbf{x}) - \log \pi_u\big(f^{-1}(\mathbf{x})\big) - \log \left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right|\right). \tag{13}$$

The inverse transformation f^{-1} from $\mathbf{x}$ to $\mathbf{u}$ can be seen as describing an implicit IAF with base density $\pi_x(\mathbf{x})$, which would define the following density over the $\mathbf{u}$ space:

$$p_u(\mathbf{u}) = \pi_x(f(\mathbf{u})) \left|\det\left(\frac{\partial f}{\partial \mathbf{u}}\right)\right|. \tag{14}$$

By making the change of variables $\mathbf{x} \mapsto \mathbf{u}$ in Equation (13) and using the definition of $p_u(\mathbf{u})$ in Equation (14) we obtain

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = \mathbb{E}_{p_u(\mathbf{u})}\left(\log \pi_x(f(\mathbf{u})) - \log \pi_u(\mathbf{u}) + \log \left|\det\left(\frac{\partial f}{\partial \mathbf{u}}\right)\right|\right) \tag{15}$$

$$= \mathbb{E}_{p_u(\mathbf{u})}(\log p_u(\mathbf{u}) - \log \pi_u(\mathbf{u})). \tag{16}$$

Equation (16) is the definition of the KL divergence from $p_u(\mathbf{u})$ to $\pi_u(\mathbf{u})$, hence

$$D_{\mathrm{KL}}(\pi_x(\mathbf{x}) \,\|\, p_x(\mathbf{x})) = D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u})). \tag{17}$$

Suppose now that we wish to fit the implicit density $p_u(\mathbf{u})$ to the base density $\pi_u(\mathbf{u})$ by minimizing the above KL. This corresponds exactly to the objective minimized when employing IAF as a recognition network in stochastic variational inference [16], where $\pi_u(\mathbf{u})$ would be the (typically intractable) posterior. The first step in stochastic variational inference would be to rewrite the expectation in Equation (16) with respect to the base distribution $\pi_x(\mathbf{x})$ used by IAF, which corresponds exactly to Equation (13). This is often referred to as the reparameterization trick [15, 28]. The second step would be to approximate Equation (13) with Monte Carlo, using samples $\{\mathbf{x}_n\}$ drawn from $\pi_x(\mathbf{x})$, as follows:

$$D_{\mathrm{KL}}(p_u(\mathbf{u}) \,\|\, \pi_u(\mathbf{u})) = \mathbb{E}_{\pi_x(\mathbf{x})}\left(\log \pi_x(\mathbf{x}) - \log \pi_u\big(f^{-1}(\mathbf{x})\big) - \log \left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right|\right) \tag{18}$$

$$\approx \frac{1}{N}\sum_n \left(\log \pi_x(\mathbf{x}_n) - \log \pi_u\big(f^{-1}(\mathbf{x}_n)\big) - \log \left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right|\right). \tag{19}$$

Using the definition of $p_x(\mathbf{x})$ in Equation (11), we can rewrite Equation (19) as

$$\frac{1}{N}\sum_n (\log \pi_x(\mathbf{x}_n) - \log p_x(\mathbf{x}_n)) = -\frac{1}{N}\sum_n \log p_x(\mathbf{x}_n) + \text{const.} \tag{20}$$

Since samples $\{\mathbf{x}_n\}$ drawn from $\pi_x(\mathbf{x})$ correspond precisely to the train data for MAF, we can recognize in Equation (20) the training objective for MAF. In conclusion, training a MAF by maximizing its total log likelihood $\sum_n \log p_x(\mathbf{x}_n)$ on train data $\{\mathbf{x}_n\}$ is equivalent to variationally training an implicit IAF with MAF's base distribution $\pi_u(\mathbf{u})$ as its target.

B Batch normalization

In our implementation of MAF, we inserted a batch normalization layer [13] between every two autoregressive layers, and between the last autoregressive layer and the base distribution. We did the

same for Real NVP (the original implementation of Real NVP also uses batch normalization layers between coupling layers [6]). The purpose of a batch normalization layer is to normalize its inputs $\mathbf{x}$ to have approximately zero mean and unit variance. In this section, we describe in full detail our implementation of batch normalization and its use as a layer in normalizing flows.

A batch normalization layer can be thought of as a transformation between two vectors of the same dimensionality. For consistency with our notation for autoregressive and coupling layers, let $\mathbf{x}$ be the vector closer to the data, and $\mathbf{u}$ be the vector closer to the base distribution. Batch normalization implements the transformation $\mathbf{x} = f(\mathbf{u})$ defined by

$$\mathbf{x} = (\mathbf{u} - \boldsymbol{\beta}) \odot \exp(-\boldsymbol{\gamma}) \odot (\mathbf{v} + \epsilon)^{\frac{1}{2}} + \mathbf{m}. \tag{21}$$

In the above, $\odot$ denotes elementwise multiplication. All other operations are to be understood elementwise. The inverse transformation f^{-1} is given by

$$\mathbf{u} = (\mathbf{x} - \mathbf{m}) \odot (\mathbf{v} + \epsilon)^{-\frac{1}{2}} \odot \exp \boldsymbol{\gamma} + \boldsymbol{\beta}, \tag{22}$$

and the absolute determinant of its Jacobian is

$$\left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right| = \exp\left(\sum_i \left(\gamma_i - \frac{1}{2}\log(v_i + \epsilon)\right)\right). \tag{23}$$

Vectors $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$ are parameters of the transformation that are learnt during training. In typical implementations of batch normalization, parameter $\boldsymbol{\gamma}$ is not exponentiated. In our implementation, we chose to exponentiate $\boldsymbol{\gamma}$ in order to ensure its positivity and simplify the expression of the log absolute determinant. Parameters $\mathbf{m}$ and $\mathbf{v}$ correspond to the mean and variance of $\mathbf{x}$ respectively. During training, we set $\mathbf{m}$ and $\mathbf{v}$ equal to the sample mean and variance of the current minibatch (we used minibatches of 100 examples). At validation and test time, we set them equal to the sample mean and variance of the entire train set. Other implementations use averages over minibatches [13] or maintain running averages during training [6]. Finally, ϵ is a hyperparameter that ensures numerical stability if any of the elements of $\mathbf{v}$ is near zero. In our experiments, we used $\epsilon = 10^{-5}$.

C Number of parameters

To get a better idea of the computational trade-offs between different model choices versus the performance gains they achieve, we compare the number of parameters for each model. We only count connection weights, as they contribute the most, and ignore biases and batch normalization parameters. We assume that masking reduces the number of connections by approximately half.

For all models, let D be the number of inputs, H be the number of units in a hidden layer and L be the number of hidden layers. We assume that all hidden layers have the same number of units (as we did in our experiments). For MAF MoG, let C be the number of components per conditional. For Real NVP and MAF, let K be the number of coupling layers/autoregressive layers respectively. Table 3 lists the number of parameters for each model.

For each extra component we add to MADE MoG, we increase the number of parameters by DH. For each extra autoregressive layer we add to MAF, we increase the number of parameters by $\frac{3}{2}DH + \frac{1}{2}(L-1)H^2$. If we have one or two hidden layers L (as we did in our experiments) and assume that D is comparable to H, the number of extra parameters in both cases is about the same. In other words, increasing flexibility by stacking has a parameter cost that is similar to adding more components to the conditionals, as long as the number of hidden layers is small.

Comparing Real NVP with MAF, we can see that Real NVP has about 1.3 to 2 times more parameters than a MAF of comparable size. Given that our experiments show that Real NVP is less flexible than a MAF of comparable size, we can conclude that MAF makes better use of its available capacity. The number of parameters of Real NVP could be reduced by tying weights between the scaling and shifting networks.

D Additional experimental details

D.1 Models

MADE, MADE MoG and each autoregressive layer in MAF is a feedforward neural network (with masked weight matrices), with L hidden layers of H hidden units each. Similarly, each coupling

Table 3: Approximate number of parameters for each model, as measured by number of connection weights. Biases and batch normalization parameters are ignored.

	# of parameters
MADE	$\frac{3}{2}DH + \frac{1}{2}(L-1)H^2$
MADE MoG	$\left(C + \frac{1}{2}\right)DH + \frac{1}{2}(L-1)H^2$
Real NVP	$2KDH + 2K(L-1)H^2$
MAF	$\frac{3}{2}KDH + \frac{1}{2}K(L-1)H^2$

layer in Real NVP contains two feedforward neural networks, one for scaling and one for shifting, each of which also has L hidden layers of H hidden units each. For each dataset, we gave a number of options for L and H (the same options where given to all models) and for each model we selected the option that performed best on the validation set. Table 4 lists the combinations of L and H that were given as options for each dataset.

In terms of nonlinearity for the hidden units, MADE, MADE MoG and MAF used rectified linear units, except for the GAS datasets where we used hyperbolic tangent units. In the coupling layer of Real NVP, we used hyberbolic tangent hidden units for the scaling network and rectified linear hidden units for the shifting network.

Table 4: Number of hidden layers L and number of hidden units H given as options for each dataset. Each combination is reported in the format $L \times H$.

POWER	GAS	HEPMASS	MINIBOONE	BSDS300	MNIST	CIFAR-10
1×100	1×100	1×512	1×512	1×512	1×1024	1×1024
2×100	2×100	2×512	2×512	2×512		2×1024
				1×1024		2×2048
				2×1024		

D.2 Datasets

In the following paragraphs, we give a brief description of the four UCI datasets (POWER, GAS, HEPMASS, MINIBOONE) and of the way they were preprocessed.

POWER. The POWER dataset [1] contains measurements of electric power consumption in a household over a period of 47 months. It is actually a time series but was treated as if each example were an i.i.d. sample from the marginal distribution. The time feature was turned into an integer for the number of minutes in the day and then uniform random noise was added to it. The date was discarded, along with the global reactive power parameter, which seemed to have many values at exactly zero, which could have caused arbitrarily large spikes in the learnt distribution. Uniform random noise was added to each feature in the interval $[0, \epsilon_i]$, where ϵ_i is large enough to ensure that with high probability there are no identical values for the i^{th} feature but small enough to not change the data values significantly.

GAS. Created by Fonollosa et al. [8], this dataset represents the readings of an array of 16 chemical sensors exposed to gas mixtures over a 12 hour period. Similarly to POWER, it is a time series but was treated as if each example were an i.i.d. sample from the marginal distribution. Only the data from the file `ethylene_CO.txt` was used, which corresponds to a mixture of ethylene and carbon monoxide. After removing strongly correlated attributes, the dimensionality was reduced to 8.

HEPMASS. Used by Baldi et al. [2], this dataset describes particle collisions in high energy physics. Half of the data are examples of particle-producing collisions (positive), whereas the rest come from a background source (negative). Here we used the positive examples from the "1000" dataset, where

the particle mass is 1000. Five features were removed because they had too many reoccurring values; values that repeat too often can result in spikes in the density and misleading results.

MINIBOONE. Used by Roe et al. [30], this dataset comes from the MiniBooNE experiment at Fermilab. Similarly to HEPMASS, it contains a number of positive examples (electron neutrinos) and a number of negative examples (muon neutrinos). Here we use the positive examples. These had some obvious outliers (11) which had values at exactly -1000 for every column and were removed. Also, seven of the features had far too high a count for a particular value, e.g. 0.0, so these were removed as well.

Table 5 lists the dimensionality and the number of train, validation and test examples for all seven datasets. The first three datasets in Table 5 were subsampled so that the product of the dimensionality and number of examples would be approximately $10\mathrm{M}$. For the four UCI datasets, 10% of the data was held out and used as test data and 10% of the remaining data was used as validation data. From the BSDS300 dataset we randomly extracted $1\mathrm{M}$ patches for training, $50\mathrm{K}$ patches for validation and $250\mathrm{K}$ patches for testing. For MNIST and CIFAR-10 we held out 10% of the train data for validation. We augmented the CIFAR-10 train set with the horizontal flips of all remaining $45\mathrm{K}$ train examples.

Table 5: Dimensionality D and number of examples N for each dataset.

		N		
	D	train	validation	test
POWER	6	1,659,917	184,435	204,928
GAS	8	852,174	94,685	105,206
HEPMASS	21	315,123	35,013	174,987
MINIBOONE	43	29,556	3,284	3,648
BSDS300	63	1,000,000	50,000	250,000
MNIST	784	50,000	10,000	10,000
CIFAR-10	3072	90,000	5,000	10,000

E Additional results

E.1 Pairwise comparison

On the MINIBOONE dataset, the model with highest average test log likelihood is MAF MoG (5). However, due to the relatively small size of this dataset, the average test log likelihoods of some other models have overlapping error bars with that of MAF MoG (5). To assess whether the differences are statistically significant, we performed a pairwise comparison, which is a more powerful statistical test. In particular, we calculated the difference in test log probability between every other model and MAF MoG (5) on each test example, and assessed whether this difference is significantly positive, which would indicate that MAF MoG (5) performs significantly better. The results of this comparison are shown in Table 6. We can see that MAF MoG (5) is significantly better than all other models except for MAF (5).

E.2 Bits per pixel

In the main text, the results for MNIST and CIFAR-10 were reported in log likelihoods in logit space, since this is the objective that the models were trained to optimize. For comparison with other results in the literature, in Table 7 we report the same results in bits per pixel. For CIFAR-10, different colour components count as different pixels (i.e. an image is thought of as having $32 \times 32 \times 3$ pixels).

In order to calculate bits per pixel, we need to transform the densities returned by a model (which refer to logit space) back to image space in the range $[0, 256]$. Let $\mathbf{x}$ be an image of D pixels in logit space and $\mathbf{z}$ be the corresponding image in $[0, 256]$ image space. The transformation from $\mathbf{z}$ to $\mathbf{x}$ is

$$\mathbf{x} = \mathrm{logit}\Big(\lambda + (1 - 2\lambda)\frac{\mathbf{z}}{256}\Big), \tag{24}$$

Table 6: Pairwise comparison results for MINIBOONE. Values correspond to average difference in log probability (in nats) from the best performing model, i.e. MAF MoG (5). Error bars correspond to 2 standard deviations. Significantly positive values indicate that MAF MoG (5) performs better.

	MINIBOONE
Gaussian	25.55 ± 0.88
MADE	3.91 ± 0.20
MADE MoG	0.59 ± 0.16
Real NVP (5)	1.87 ± 0.16
Real NVP (10)	2.15 ± 0.21
MAF (5)	$\mathbf{0.07 \pm 0.11}$
MAF (10)	0.55 ± 0.12
MAF MoG (5)	$\mathbf{0.00 \pm 0.00}$

Table 7: Bits per pixel for conditional density estimation (lower is better). The best performing model for each dataset is shown in bold. Error bars correspond to 2 standard deviations.

	MNIST		CIFAR-10	
	unconditional	conditional	unconditional	conditional
Gaussian	2.01 ± 0.01	1.97 ± 0.01	4.63 ± 0.01	4.79 ± 0.02
MADE	2.04 ± 0.01	2.00 ± 0.01	5.67 ± 0.01	5.65 ± 0.01
MADE MoG	$\mathbf{1.41 \pm 0.01}$	$\mathbf{1.39 \pm 0.01}$	5.93 ± 0.01	5.80 ± 0.01
Real NVP (5)	1.93 ± 0.01	1.94 ± 0.01	4.53 ± 0.01	4.50 ± 0.01
Real NVP (10)	2.02 ± 0.02	2.02 ± 0.08	4.54 ± 0.01	4.58 ± 0.01
MAF (5)	1.89 ± 0.01	1.89 ± 0.01*	4.36 ± 0.01	4.34 ± 0.01*
MAF (10)	1.91 ± 0.01	1.92 ± 0.01*	$\mathbf{4.31 \pm 0.01}$	$\mathbf{4.30 \pm 0.01}$*
MAF MoG (5)	1.52 ± 0.01	1.51 ± 0.01	4.37 ± 0.01	4.36 ± 0.01

where $\lambda = 10^{-6}$ for MNIST and $\lambda = 0.05$ for CIFAR-10. If $p(\mathbf{x})$ is the density in logit space as returned by the model, using the above transformation the density of $\mathbf{z}$ can be calculated as

$$p_z(\mathbf{z}) = p(\mathbf{x}) \left(\frac{1-2\lambda}{256}\right)^D \left(\prod_i \sigma(x_i)(1-\sigma(x_i))\right)^{-1}, \tag{25}$$

where $\sigma(\cdot)$ is the logistic sigmoid function. From that, we can calculate the bits per pixel $b(\mathbf{x})$ of image $\mathbf{x}$ as follows:

$$b(\mathbf{x}) = -\frac{\log_2 p_z(\mathbf{z})}{D} \tag{26}$$

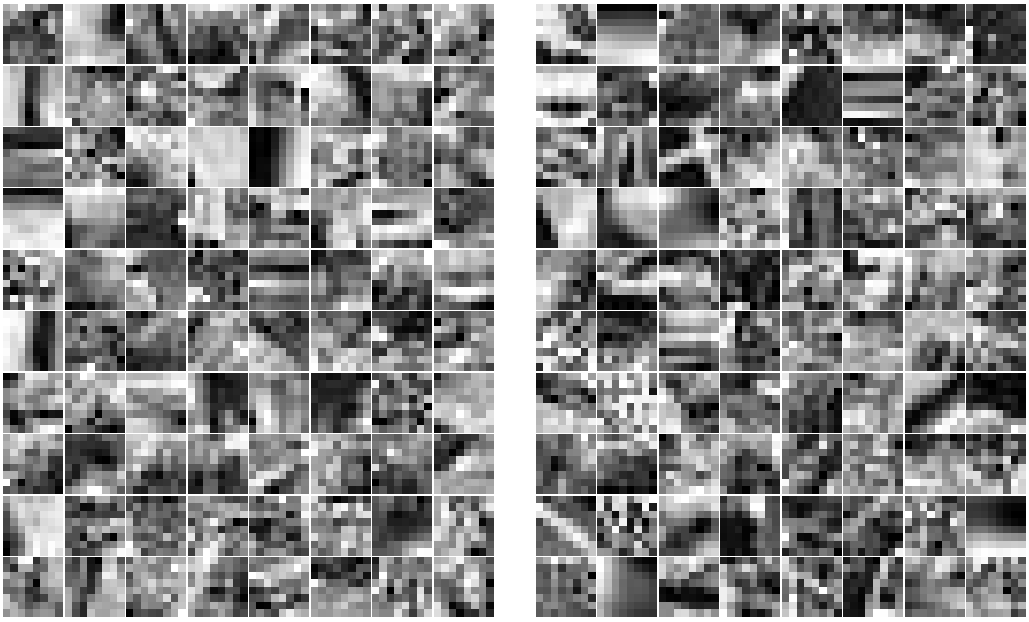
$$= -\frac{\log p(\mathbf{x})}{D \log 2} - \log_2(1-2\lambda) + 8 + \frac{1}{D}\sum_i (\log_2 \sigma(x_i) + \log_2(1-\sigma(x_i))). \tag{27}$$

The above equation was used to convert between the average log likelihoods reported in the main text and the results of Table 7.

E.3 Generated images

Figures 2, 3 and 4 show generated images and real examples for BSDS300, MNIST and CIFAR-10 respectively. Images were generated by MAF MoG (5) for BSDS300, conditional MAF (5) for MNIST, and conditional MAF (10) for CIFAR-10.

The BSDS300 generated images are visually indistinguishable from the real ones. For MNIST and CIFAR-10, generated images lack the fidelity produced by modern image-based generative approaches, such as RealNVP [6] or PixelCNN++ [31]. This is because our version of MAF has



(a) Generated images (b) Real images

Figure 2: Generated and real images from BSDS300.

no knowledge about image structure, as it was designed for general-purpose density estimation and not for realistic-looking image synthesis. However, if the latter is desired, it would be possible to incorporate image modelling techniques in the design of MAF (such as convolutions or a multi-scale architecture as used by Real NVP [6]) in order to improve quality of generated images.

References

[1] Individual household electric power consumption data set. `http://archive.ics.uci.edu/ml/datasets/Individual+household+electric+power+consumption`. Accessed on 15 May 2017.

[2] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson. Parameterized machine learning for high-energy physics. arXiv:1601.07913, 2016.

[3] J. Ballé, V. Laparra, and E. P. Simoncelli. Density modeling of images using a generalized normalization transformation. *Proceedings of the 4nd International Conference on Learning Representations*, 2016.

[4] S. S. Chen and R. A. Gopinath. Gaussianization. *Advances in Neural Information Processing Systems 13*, pages 423–429, 2001.

[5] L. Dinh, D. Krueger, and Y. Bengio. NICE: Non-linear Independent Components Estimation. arXiv:1410.8516, 2014.

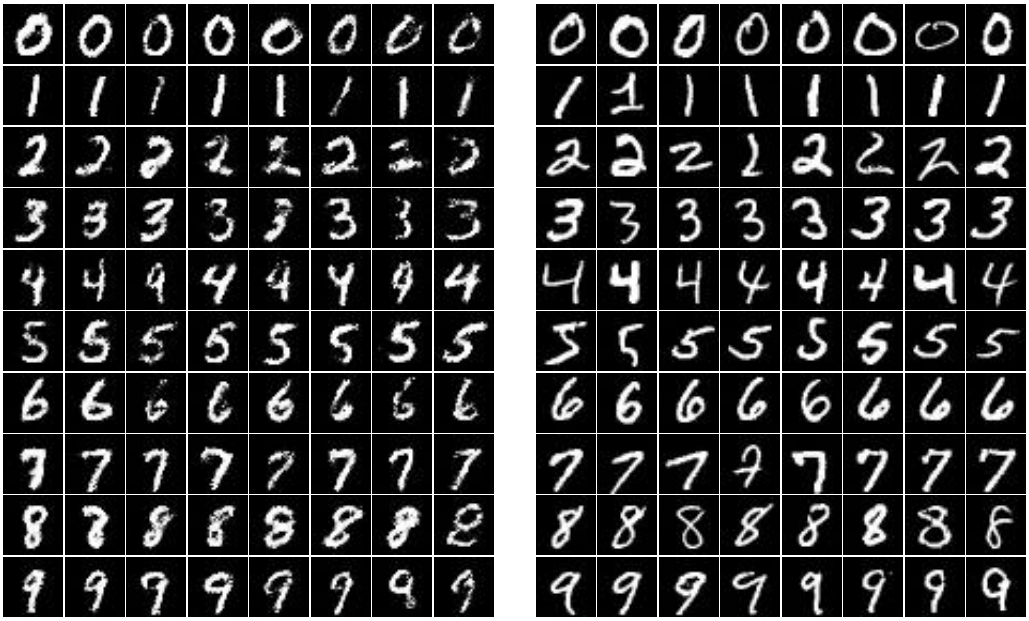
[6] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[7] Y. Fan, D. J. Nott, and S. A. Sisson. Approximate Bayesian computation via regression density estimation. *Stat*, 2(1):34–48, 2013.

[8] J. Fonollosa, S. Sheik, R. Huerta, and S. Marco. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical*, 215:618–629, 2015.

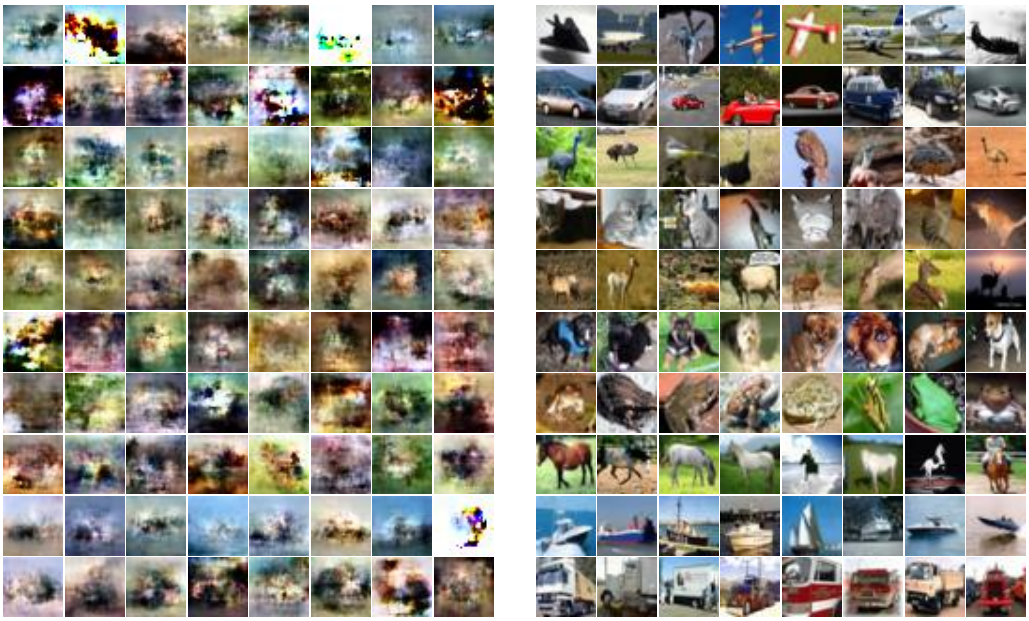
[9] M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked Autoencoder for Distribution Estimation. *Proceedings of the 32nd International Conference on Machine Learning*, pages 881–889, 2015.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.



(a) Generated images (b) Real images

Figure 3: Class-conditional generated and real images from MNIST. Rows are different classes. Generated images are sorted by decreasing log likelihood from left to right.



(a) Generated images (b) Real images

Figure 4: Class-conditional generated and real images from CIFAR-10. Rows are different classes. Generated images are sorted by decreasing log likelihood from left to right.

[11] S. Gu, Z. Ghahramani, and R. E. Turner. Neural adaptive sequential Monte Carlo. *Advances in Neural Information Processing Systems 28*, pages 2629–2637, 2015.

[12] G. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[13] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning*, pages 448–456, 2015.

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[15] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

[16] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with Inverse Autoregressive Flow. *Advances in Neural Information Processing Systems 29*, pages 4743–4751, 2016.

[17] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[18] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. Mansinghka. Picture: A probabilistic programming language for scene perception. *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4390–4399, 2015.

[19] T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

[20] Y. LeCun, C. Cortes, and C. J. C. Burges. The MNIST database of handwritten digits. URL `http://yann.lecun.com/exdb/mnist/`.

[21] M. Lichman. UCI machine learning repository, 2013. URL `http://archive.ics.uci.edu/ml`.

[22] G. Loaiza-Ganem, Y. Gao, and J. P. Cunningham. Maximum entropy flow networks. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[23] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. pages 416–423, 2001.

[24] B. Paige and F. Wood. Inference networks for sequential Monte Carlo in graphical models. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[25] G. Papamakarios and I. Murray. Distilling intractable generative models, 2015. Probabilistic Integration Workshop at Neural Information Processing Systems 28.

[26] G. Papamakarios and I. Murray. Fast ϵ-free inference of simulation models with Bayesian conditional density estimation. *Advances in Neural Information Processing Systems 29*, 2016.

[27] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538, 2015.

[28] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *Proceedings of the 31st International Conference on Machine Learning*, pages 1278–1286, 2014.

[29] O. Rippel and R. P. Adams. High-dimensional probability estimation with deep density models. arXiv:1302.5125, 2013.

[30] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor. Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 543(2–3):577–584, 2005.

[31] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. arXiv:1701.05517, 2017.

[32] Y. Tang, R. Salakhutdinov, and G. Hinton. Deep mixtures of factor analysers. *Proceedings of the 29th International Conference on Machine Learning*, pages 505–512, 2012.

[33] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. arXiv:1605.02688, 2016.

[34] L. Theis and M. Bethge. Generative image modeling using spatial LSTMs. *Advances in Neural Information Processing Systems 28*, pages 1927–1935, 2015.

[35] L. Theis, A. van den Oord, and M. Bethge. A note on the evaluation of generative models. *Proceedings of the 4nd International Conference on Learning Representations*, 2016.

[36] B. Uria, I. Murray, and H. Larochelle. RNADE: The real-valued neural autoregressive density-estimator. *Advances in Neural Information Processing Systems 26*, pages 2175–2183, 2013.

[37] B. Uria, I. Murray, and H. Larochelle. A deep and tractable density estimator. *Proceedings of the 31st International Conference on Machine Learning*, pages 467–475, 2014.

[38] B. Uria, I. Murray, S. Renals, C. Valentini-Botinhao, and J. Bridle. Modelling acoustic feature dependencies with artificial neural networks: Trajectory-RNADE. *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4465–4469, 2015.

[39] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.

[40] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. arXiv:1609.03499, 2016.

[41] A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with PixelCNN decoders. *Advances in Neural Information Processing Systems 29*, pages 4790–4798, 2016.

[42] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *Proceedings of the 33rd International Conference on Machine Learning*, pages 1747–1756, 2016.

[43] D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. *Proceedings of the 13rd International Conference on Computer Vision*, pages 479–486, 2011.

2.4 Contribution and impact

The paper *Masked Autoregressive Flow for Density Estimation* showed that we can increase the flexibility of simple neural density estimators by explicitly modelling their internal randomness. By reparameterizing a density model in terms of its internal randomness, we obtain an invertible transformation that can be composed into a normalizing flow. The resulting model can be more expressive than the original, while it remains tractable to train, evaluate and sample from. *Masked Autoregressive Flow* is a specific implementation of this idea that uses masked autoregressive models with Gaussian conditionals as building blocks.

In addition to introducing a new density estimator, the paper contributed to the understanding of the relationship between MADE, MAF, IAF and Real NVP, and clarified the computational tradeoffs involved with using these models for density estimation and variational inference. Specifically, the paper explained the following relationships:

(i) A MAF with one layer is a MADE with Gaussian conditionals.

(ii) An IAF is a MAF with its transformation inverted (and vice versa).

(iii) A Real NVP is a MAF/IAF with one autoregressive step instead of D autoregressive steps.

(iv) Fitting a MAF to the training data by maximum likelihood can be viewed as fitting an implicit IAF to the base density by stochastic variational inference.

Finally, the paper clarified the following computational tradeoffs:

(i) MAF is fast to evaluate but slow to sample from.

(ii) IAF is slow to evaluate but fast to sample from.

(iii) Real NVP is fast to both evaluate and sample from, at the cost of decreased flexibility compared to MAF/IAF.

According to Google Scholar, the paper has received 66 citations as of April 2019. MAF has been used as a prior and/or decoder for variational autoencoders (Alemi et al., 2018; Bauer and Mnih, 2019; Choi et al., 2019; Dillon et al., 2017; Tran et al., 2018; Vikram et al., 2019), for modelling state-action pairs in imitation learning (Schroecker et al., 2019), and for estimating likelihoods or likelihood ratios in likelihood-free inference (Brehmer et al., 2018c; Papamakarios et al., 2019). The five datasets we used for our experiments on unconditional density estimation (namely POWER, GAS, HEPMASS, MINIBOONE and BSDS300) have been made available online (Papamakarios, 2018) and have been used by other researchers as density-estimation benchmarks (De Cao et al., 2019; Grathwohl et al., 2018; Huang et al., 2018; Li and Grathwohl, 2018; Nash and Durkan, 2019; Oliva et al., 2018). Finally, MAF has been implemented as part of *TensorFlow Probability* (Dillon et al., 2017), a software library for probabilistic modelling and inference which forms part of TensorFlow (Abadi et al., 2015).

There are two ways in which MAF is limited. First, MAF can be slow to sample from in high dimensions, as the computational cost of generating D-dimensional data from MAF scales linearly with D. This is true in general for autoregressive models, such as WaveNet (van den Oord et al., 2016a) or PixelCNN (Salimans et al., 2017; van den Oord et al., 2016b). Slow sampling limits the applicability of MAF as a neural sampler or as a variational posterior. Second, it is still unknown whether MAF is a universal density approximator, i.e. whether it can model any well-behaved density arbitrarily well given enough layers and hidden units. In the next section, I will review advances in normalizing flows since the publication of the paper, and I will discuss further the tradeoffs between efficiency and expressivity in existing models.

2.5 Further advances in normalizing flows

Since the publication of the paper *Masked Autoregressive Flow for Density Estimation*, there has been a lot of research interest in normalizing flows. In this section, I review advances in normalizing flows after the publication of the paper, and I show how the various approaches are related to each other. Some of the advances are based on Masked Autoregressive Flow, and others are independent threads of research.

2.5.1 Non-affine autoregressive layers

MAF, IAF and Real NVP are all composed of *affine autoregressive layers*, i.e. autoregressive layers where each variable is scaled and shifted as a function of previous variables. (Since the coupling layers used by Real NVP are special cases of autoregressive layers, I won't make a distinction between the two from now on.) As we discussed in section 2.3, an affine autoregressive layer transforms each noise variable u_i into a data variable x_i as follows:

$$x_i = \alpha_i u_i + \beta_i, \tag{2.32}$$

where α_i and β_i are functions of $\mathbf{x}_{1:i-1}$ or $\mathbf{u}_{1:i-1}$. Restricting the transformation from u_i to x_i to be affine allowed us to invert it and compute the determinant of its Jacobian efficiently.

However, one could increase the expressivity of an autoregressive layer by allowing more general transformations from u_i to x_i of the form:

$$x_i = g_{\boldsymbol{\psi}_i}(u_i), \tag{2.33}$$

where $\boldsymbol{\psi}_i$ is a function of $\mathbf{x}_{1:i-1}$ or $\mathbf{u}_{1:i-1}$ that parameterizes the transformation. As long as $g_{\boldsymbol{\psi}_i}$ is taken to be smooth and invertible, the resulting flow is a *non-affine autoregressive flow*. The absolute determinant of the Jacobian of such a flow is:

$$\left|\det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}}\right)\right| = \prod_i \left|\frac{\partial g_{\boldsymbol{\psi}_i}}{\partial u_i}\right|^{-1}. \tag{2.34}$$

Neural autoregressive flows

Neural autoregressive flows (Huang et al., 2018) are non-affine autoregressive flows that use monotonically-increasing neural networks to parameterize $g_{\boldsymbol{\psi}_i}$. A feedforward neural network with one input u_i and one output x_i can be made monotonically increasing if (a) all its activation functions are monotonically increasing (sigmoid or leaky-ReLU activation functions have this property), and (b) all its weights are strictly positive. Huang et al. (2018) propose neural architectures that follow this principle, termed *deep sigmoidal flows* and *deep dense sigmoidal flows*. The derivative of $g_{\boldsymbol{\psi}_i}$ with respect to its input (needed for the computation of the Jacobian determinant above) can be obtained by automatic differentiation. A special case of a neural autoregressive flow is *Flow++* (Ho et al., 2019), which parameterizes $g_{\boldsymbol{\psi}_i}$ as a mixture of logistic CDFs, and is equivalent to a neural network with positive weights and one hidden layer of logistic-sigmoid units.

The advantage of neural autoregressive flows is their expressivity. Huang et al. (2018) show that with a sufficiently flexible transformation $g_{\boldsymbol{\psi}_i}$, a single neural autoregressive layer can approximate any well-behaved density arbitrarily well. This is because if $g_{\boldsymbol{\psi}_i}$ becomes equal to the inverse CDF of the conditional $p(x_i \,|\, \mathbf{x}_{1:i-1})$, the neural autoregressive layer transforms the joint density $p(\mathbf{x})$ into a uniform density in the unit cube (Hyvärinen and Pajunen, 1999).

The disadvantage of neural autoregressive flows is that in general they are not analytically invertible. That is, even though the inverse of $g_{\boldsymbol{\psi}_i}$ exists, it's not always available in closed form. In order to invert the flow, one would have to resort to numerical methods. A neural autoregressive flow can still be used to estimate densities if it is taken to parameterize the transformation from $\mathbf{x}$ to $\mathbf{u}$, but if the transformation from $\mathbf{u}$ to $\mathbf{x}$ is not available analytically, it would not be possible to sample from the trained model efficiently.

Non-linear squared flow

If we take $g_{\boldsymbol{\psi}_i}$ to be non-affine but restrict it to have an analytic inverse, we would have a non-affine autoregressive flow that we could sample from. An example is the *non-linear squared flow* (Ziegler and Rush, 2019), which adds an inverse-quadratic perturbation to the affine transformation as follows:

$$g_{\boldsymbol{\psi}_i}(u_i) = \alpha_i u_i + \beta_i + \frac{\gamma_i}{1 + (\delta_i u_i + \epsilon_i)^2} \quad \text{where} \quad \boldsymbol{\psi}_i = \{\alpha_i, \beta_i, \gamma_i, \delta_i, \epsilon_i\}\,. \tag{2.35}$$

The above transformation is not generally invertible, but it can be made monotonically increasing if we restrict $\alpha_i > \frac{9}{8\sqrt{3}} |\gamma_i| \, \delta_i$ and $\delta_i > 0$. For $\gamma_i = 0$, the non-linear squared flow reduces to an affine autoregressive flow. Given x_i, the equation $x_i = g_{\boldsymbol{\psi}_i}(u_i)$ is a cubic polynomial with respect to u_i, so it can be solved analytically. The above transformation is more expressive than an affine transformation, but not as expressive as a general neural autoregressive flow.

Piecewise-polynomial autoregressive flows

Another approach to creating non-affine but analytically invertible autoregressive flows is to parameterize $g_{\boldsymbol{\psi}_i}$ as a piecewise-linear or piecewise-quadratic monotonically-increasing function (Müller et al., 2018). In this case, the parameters $\boldsymbol{\psi}_i$ correspond to the locations of the segments and their shape (i.e. slope and curvature). In order to invert $g_{\boldsymbol{\psi}_i}$ for a given x_i, one would need to first identify which segment this x_i corresponds to (which can be done by binary search since the segments are sorted), and then invert that segment (which is easy for a linear or quadratic segment). The more segments we have, the more flexible the transformation becomes.

2.5.2 Invertible convolutional layers

If we are interested in modelling image data, we may want to design a flow that contains invertible convolutional layers. For this discussion, we will assume that the data is an image of shape $H \times W \times C$, where H is the height, W is the width, and C is the number of channels. A convolutional layer transforms noise of shape $H \times W \times C$ into data via a convolution with filter k. Let $\mathbf{x}$ and $\mathbf{u}$ represent the vectorized image and noise respectively. Since convolution is a linear operation, we can write it as the following matrix multiplication:

$$\mathbf{x} = \mathbf{W}_k \, \mathbf{u}, \tag{2.36}$$

where $\mathbf{W}_k$ is a matrix of shape $HWC \times HWC$ whose entries depend on the filter k. If $\mathbf{W}_k$ is invertible then the convolution is invertible, and its Jacobian has absolute determinant:

$$\left| \det\left(\frac{\partial f^{-1}}{\partial \mathbf{x}} \right) \right| = \left| \det(\mathbf{W}_k) \right|^{-1}. \tag{2.37}$$

Nonetheless, naively inverting $\mathbf{W}_k$ or calculating its determinant has a cost of $\mathcal{O}\big((HWC)^3\big)$, so more scalable solutions have to be found in practice.

Invertible 1×1 convolutions and Glow

Kingma and Dhariwal (2018) introduced *invertible* 1×1 *convolutions* in their model *Glow*. An invertible 1×1 convolution is essentially a linear transformation where each pixel of size C (with one value for each channel) is multiplied by the same matrix $\mathbf{V}$ of shape $C \times C$. The equivalent matrix $\mathbf{W}_k$ can be obtained by:

$$\mathbf{W}_k = \mathbf{V} \otimes \mathbf{I}, \tag{2.38}$$

where $\mathbf{I}$ is the identity matrix of shape $HW \times HW$ and $\otimes$ is the Kronecker product. The inverse and determinant of $\mathbf{W}_k$ have a cost of $\mathcal{O}\big(C^3\big)$, which may not be prohibitive for moderate C. To further reduce the cost, Kingma and Dhariwal (2018) suggest parameterizing $\mathbf{V}$ as follows:

$$\mathbf{V} = \mathbf{PLU}, \tag{2.39}$$

where $\mathbf{P}$ is a fixed permutation matrix, $\mathbf{L}$ is a lower triangular matrix with ones in its diagonal, and $\mathbf{U}$ is an upper triangular matrix. In that case, the absolute determinant of $\mathbf{W}_k$ becomes:

$$|\det(\mathbf{W}_k)| = HW \prod_c |U_{cc}|\,, \tag{2.40}$$

where U_{cc} is the c-th element of $\mathbf{U}$'s diagonal. If we further restrict every U_{cc} to be positive, we guarantee that the transformation is always invertible. In addition to modelling images, invertible 1×1 convolutions have been used in modelling audio by *WaveGlow* (Prenger et al., 2018) and *FloWaveNet* (Kim et al., 2018).

Autoregressive and emerging convolutions

One way of obtaining scalable invertible convolutions without restricting the receptive field to be 1×1 is via *autoregressive convolutions* (Hoogeboom et al., 2019). In an autoregressive convolution, pixels are assumed to be ordered, and part of the filter is zeroed out so that output pixel i only depends on input pixels 1 to $i - 1$. An autoregressive convolution corresponds to a triangular matrix $\mathbf{W}_k$, and hence its determinant can be calculated at a cost of $\mathcal{O}(HWC)$. A convolution that is not restricted to be autoregressive can be obtained by composing an autoregressive convolution whose matrix $\mathbf{W}_{k_1}$ is upper triangular with an autoregressive convolution whose matrix $\mathbf{W}_{k_2}$ is lower triangular; the result is equivalent to a non-autoregressive convolution with matrix $\mathbf{W}_{k_2}\mathbf{W}_{k_1}$. This is analogous to parameterizing an LU decomposition of the convolution matrix, and is termed *emerging convolution* (Hoogeboom et al., 2019).

Periodic convolutions in the Fourier domain

Finally, another way of scaling up invertible convolutions is via the Fourier domain. According to the convolution theorem, the convolution between a filter k and a signal u is equal to:

$$k * u = \mathcal{F}^{-1}(\mathcal{F}(k)\,\mathcal{F}(u)), \tag{2.41}$$

where $\mathcal{F}$ is the Fourier transform and $\mathcal{F}^{-1}$ is its inverse. Since the Fourier transform is a unitary linear operator, its discrete version corresponds to multiplication with a particular unitary matrix $\mathbf{F}$. Hence, the convolution of a discrete signal $\mathbf{u}$ can be written in vectorized form as:

$$\mathbf{W}_k\,\mathbf{u} = \mathbf{F}^T(\mathbf{Fk} \odot \mathbf{Fu}) = \left(\mathbf{F}^T\mathbf{D}_k\,\mathbf{F}\right)\mathbf{u} \tag{2.42}$$

where $\odot$ is elementwise multiplication, and $\mathbf{D}_k$ is a diagonal matrix whose diagonal is $\mathbf{Fk}$. Since the absolute determinant of $\mathbf{F}$ is 1, the absolute determinant of $\mathbf{W}_k$ is:

$$|\det(\mathbf{W}_k)| = \prod_i |D_{k,ii}|\,, \tag{2.43}$$

where $D_{k,ii}$ is the i-th element of the filter k expressed in the Fourier domain.

In a typical convolution layer with C filters and C input channels, we perform a total of C^2 convolutions (one for each combination of filter and input-channel), and the resulting C^2 output maps are summed across input channels to obtain C output channels. If we express the entire convolution layer in the Fourier domain as we did above, we obtain a block-diagonal matrix $\mathbf{D}_k$ of shape $HWC \times HWC$ whose diagonal contains HW matrices of shape $C \times C$ (Hoogeboom et al., 2019). Hence, calculating the determinant of a convolution layer using its Fourier representation has a cost of $\mathcal{O}\big(HWC^3\big)$, which may be acceptable for moderate C. This type of convolution layer is termed *periodic convolution* (Hoogeboom et al., 2019).

2.5.3 Invertible residual layers

A residual layer (He et al., 2016) is a transformation of the following form:

$$\mathbf{x} = f(\mathbf{u}) = \mathbf{u} + g(\mathbf{u}). \tag{2.44}$$

Residual layers are designed to avoid vanishing gradients in deep neural networks. Since the Jacobian of a residual layer is:

$$\frac{\partial f}{\partial \mathbf{u}} = \mathbf{I} + \frac{\partial g}{\partial \mathbf{u}}, \tag{2.45}$$

the propagated gradient doesn't vanish even if the Jacobian of g does.

Sylvester flow

Residual networks are not generally invertible, but can be made to be if g is restricted accordingly. One such example is the *Sylvester flow* (van den Berg et al., 2018), where g is taken to be:

$$g(\mathbf{u}) = \mathbf{Q}\mathbf{R}\, h\Big(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{u} + \mathbf{b}\Big). \tag{2.46}$$

In the above, $\mathbf{Q}$ is a $D \times M$ matrix whose columns form an orthonormal basis, $\mathbf{R}$ and $\tilde{\mathbf{R}}$ are $M \times M$ upper-triangular matrices, $\mathbf{b}$ is an M-dimensional bias, $h(\cdot)$ is a smooth activation function applied elementwise, and $M \leq D$. The above transformation can be thought of as a feedforward neural network with one hidden layer of M units, whose weight matrices have been parameterized in a particular way.

Theorem 2 of van den Berg et al. (2018) gives sufficient conditions for the invertibility of the Sylvester flow. However, their proof requires $\tilde{\mathbf{R}}$ to be invertible, which as I show here is not necessary. In theorem 2.1 below, I give a more succinct proof of the invertibility of the Sylvester flow, which doesn't require the invertibility of $\tilde{\mathbf{R}}$.

Theorem 2.1 (Invertibility of the Sylvester flow)**.** The Sylvester flow is invertible if $h(\cdot)$ is monotonically increasing with bounded derivative (e.g. a sigmoid activation function has this property), and if for all m we have:

$$\tilde{R}_{mm}R_{mm} > -\frac{1}{\sup_z h'(z)}. \tag{2.47}$$

Proof. Using the *matrix-determinant lemma*, the determinant of the Jacobian of f can be written as:

$$\det\left(\frac{\partial f}{\partial \mathbf{u}}\right) = \det\Big(\mathbf{I} + \mathbf{A}\tilde{\mathbf{R}}\mathbf{R}\Big), \tag{2.48}$$

where $\mathbf{A}$ is an $M \times M$ diagonal matrix whose diagonal is $h'\Big(\tilde{\mathbf{R}}\mathbf{Q}^T\mathbf{u} + \mathbf{b}\Big)$. Since $\mathbf{I} + \mathbf{A}\tilde{\mathbf{R}}\mathbf{R}$ is an $M \times M$ upper-triangular matrix, its determinant is the product of its diagonal elements, hence:

$$\det\left(\frac{\partial f}{\partial \mathbf{u}}\right) = \prod_m \Big(1 + A_{mm}\tilde{R}_{mm}R_{mm}\Big). \tag{2.49}$$

Condition (2.47) ensures that the Jacobian determinant is positive everywhere. Hence, from the *inverse-function theorem* it follows that f is invertible. □

Sylvester flows are related to other normalizing flows. For $M = 1$, the Sylvester flow becomes a special case of the *planar flow* (Rezende and Mohamed, 2015). Furthermore, if $M = D$, $\mathbf{Q}$ is taken to be a reverse-permutation matrix and $\mathbf{R}$ is strictly upper triangular, g becomes a MADE (Germain et al., 2015) with one hidden layer of D units. In that case, the Sylvester flow becomes a special case of the *Inverse Autoregressive Flow* (Kingma et al., 2016), where the scaling factor is 1 and the shifting factor is $g(\mathbf{u})$.

Even though the Sylvester flow is invertible and has a tractable Jacobian, its inverse is not available analytically. This means that the Sylvester flow can calculate efficiently only the density of its own samples, so it can be used as a variational posterior. Alternatively, if we take it to parameterize the transformation from $\mathbf{x}$ to $\mathbf{u}$, we can use it as a density estimator but we won't be able to sample from it efficiently.

Contractive residual layers and iResNet

In general, a residual layer is invertible if g is a *contraction*, i.e. its Lipschitz constant is less than 1. A residual layer with this property is termed an *iResNet* (Behrmann et al., 2018). To see why an iResNet is invertible, fix a value for $\mathbf{x}$ and consider the sequence $\mathbf{u}_1, \mathbf{u}_2, \ldots$ obtained by:

$$\mathbf{u}_{k+1} = \mathbf{x} - g(\mathbf{u}_k). \tag{2.50}$$

The map $\mathbf{u}_k \mapsto \mathbf{u}_{k+1}$ is a contraction, so by the *Banach fixed-point theorem* it follows that the sequence converges for any choice of $\mathbf{u}_1$ to the same fixed point $\mathbf{u}_\infty$. That fixed point is the unique value that satisfies $\mathbf{x} = f(\mathbf{u}_\infty)$, which proves the invertibility of f.

One way to construct an iResNet is to parameterize g to be a feedforward neural network with contractive activation functions (such as sigmoids or ReLUs), and with weight matrices of spectral norm less than 1. Even though its inverse is not analytically available in general, an iResNet can be numerically inverted using the iterative procedure of equation (2.50).

Directly calculating the Jacobian determinant of an iResNet costs $\mathcal{O}\big(D^3\big)$. Alternatively, following Behrmann et al. (2018), the log absolute determinant of the Jacobian can be first expanded into a power series as follows:

$$\log\left|\det\left(\frac{\partial f}{\partial \mathbf{u}}\right)\right| = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \operatorname{tr}\big(\mathbf{J}_g^k\big) \quad \text{where} \quad \mathbf{J}_g = \frac{\partial g}{\partial \mathbf{u}}, \tag{2.51}$$

and then it can be approximated by truncating the power series at a desired accuracy. Further, $\operatorname{tr}\big(\mathbf{J}_g^k\big)$ can be approximated by the following unbiased stochastic estimator, known as the *Hutchinson estimator* (Hutchinson, 1990):

$$\operatorname{tr}\big(\mathbf{J}_g^k\big) \approx \mathbf{v}^T\mathbf{J}_g^k\mathbf{v} \quad \text{where} \quad \mathbf{v} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{2.52}$$

Calculating $\mathbf{v}^T\mathbf{J}_g^k\mathbf{v}$ doesn't require explicitly computing the Jacobian, as it can be done by backpropagating through g a total of k times, once for each Jacobian-vector product.

The main advantage of iResNets over other types of invertible residual networks is the flexibility in constructing g. Unlike Sylvester flows where g is restricted to one hidden layer and to no more than D hidden units, an iResNet can have any number of layers and hidden units. However, unlike other normalizing flows, it is expensive both to sample and to calculate exact densities under an iResNet, which limits its applicability in practice.

2.5.4 Infinitesimal flows

So far I have discussed normalizing flows consisting of a fixed number of layers. We can imagine a flow where the number of layers grows larger and larger, but at the same time the effect of each layer becomes smaller and smaller. In the limit of infinitely many layers each of which has an infinitesimal effect, we obtain an flow where $\mathbf{u}$ is transformed continuously, rather than in discrete steps. We call such flows *infinitesimal flows*.

Deep diffeomorphic flow

One type of infinitesimal flow is the *deep diffeomorphic flow* (Salman et al., 2018). We start with a residual layer mapping $\mathbf{u}_t$ to $\mathbf{u}_{t+1}$:

$$\mathbf{u}_{t+1} = \mathbf{u}_t + g_t(\mathbf{u}_t), \tag{2.53}$$

and then extend it to a variable-sized step as follows:

$$\mathbf{u}_{t+dt} = \mathbf{u}_t + dt\, g_t(\mathbf{u}_t). \tag{2.54}$$

In the limit $dt \to 0$, we obtain the following ordinary differential equation:

$$\frac{d\mathbf{u}_t}{dt} = g_t(\mathbf{u}_t). \tag{2.55}$$

In the above ODE, we can interpret g_t as a time-varying velocity field. For small enough dt, we can interpret a residual layer of the form of equation (2.54) as the *Euler integrator* of this ODE. In that case, we can approximately invert the flow by running the integrator backwards:

$$\mathbf{u}_t \approx \mathbf{u}_{t+dt} - dt\, g_t(\mathbf{u}_{t+dt}), \tag{2.56}$$

which becomes exact for $dt \to 0$. We can also think of a diffeomorphic flow as a special case of an iResNet, where dt is small enough such that the transformation is contractive. In that case, each step of running the integrator backwards corresponds to a single iteration of the iResNet-inversion algorithm, where $\mathbf{u}_t$ is initialized with $\mathbf{u}_{t+dt}$, the value to be inverted at each step.

Since the deep diffeomorphic flow is a special case of an iResNet, calculating its Jacobian determinant can be done similarly. Using a small (but not infinitesimal) dt, we begin by writing the log absolute determinant of each step of the Euler integrator as a power series:

$$\log\left|\det\left(\frac{\partial \mathbf{u}_{t+dt}}{\partial \mathbf{u}_t}\right)\right| = \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} (dt)^k \operatorname{tr}\left(\mathbf{J}_{g_t}^k\right) \quad \text{where} \quad \mathbf{J}_{g_t} = \frac{\partial g_t}{\partial \mathbf{u}_t}, \tag{2.57}$$

and then approximate it by truncating the power series at a desired level of accuracy. The smaller we take dt the more we can afford to truncate. Finally, we can estimate $\operatorname{tr}\left(\mathbf{J}_{g_t}^k\right)$ using the Hutchinson estimator in equation (2.52).

The main drawback of the deep diffeomorphic flow is that inverting it and computing its Jacobian determinant are approximate operations that can introduce approximation error. To make the approximation more accurate, one needs to reduce dt, which in turn can make the flow prohibitively deep, since the number of layers scales as $\mathcal{O}(1/dt)$.

Neural ODEs and FFJORD

The drawbacks of the deep diffeomorphic flow stem from the fact that (a) it backpropagates through the integrator, (b) the Euler integrator it uses is not exactly invertible, and (c) the calculation of the Jacobian determinant is approximate. A different approach that avoids these issues is *Neural ODEs* (Chen et al., 2018). Instead of using the Euler integrator and backpropagating through it, Neural ODEs defines an additional ODE that describes the evolution of the gradient of a loss as it backpropagates through the flow. Hence, instead of backpropagating through the integrator in order to train the flow, the gradients with respect to its parameters can be obtained as the solution to this additional ODE. Both the ODE for the forward pass and the ODE for the backward pass can be solved with any integrator, which allows one to use an integrator that is exactly invertible. Furthermore, the integrator can choose the number of integration steps (or layers of the flow) adaptively.

Instead of approximating the Jacobian determinant by truncating its power series, we can define directly how the log density evolves through the flow via a third ODE. We start by rewriting the power series of the log absolute determinant as follows:

$$\log \left| \det \left(\frac{\partial \mathbf{u}_{t+dt}}{\partial \mathbf{u}_t} \right) \right| = dt \ \mathrm{tr}(\mathbf{J}_{g_t}) + \mathcal{O}\big(dt^2\big). \tag{2.58}$$

Using the above, we can express the evolution of the log density from t to $t + dt$ as:

$$\log q_{t+dt}(\mathbf{u}_{t+dt}) = \log q_t(\mathbf{u}_t) - dt \ \mathrm{tr}(\mathbf{J}_{g_t}) + \mathcal{O}\big(dt^2\big). \tag{2.59}$$

Taking the limit $dt \to 0$, we obtain:

$$\frac{d}{dt} \log q_t(\mathbf{u}_t) = -\mathrm{tr}(\mathbf{J}_{g_t}) = -\nabla \cdot g_t(\mathbf{u}_t). \tag{2.60}$$

The above ODE is a special case of the *Fokker–Planck equation* where the diffusion is zero. As before, $\mathrm{tr}(\mathbf{J}_{g_t})$ can be approximated using the Hutchinson estimator. The above ODE can be solved together with the ODEs for forward and backward propagation through the flow with the integrator of our choice. The resulting flow is termed *FFJORD* (Grathwohl et al., 2018). A special case of FFJORD is the *Monge–Ampère flow* (Zhang et al., 2018), which takes g_t to be the gradient of a scalar field (in which case g_t has zero rotation everywhere).

The advantage of Neural ODEs and FFJORD is their flexibility. Unlike other normalizing flows where the architecture has to be restricted in some way or another, g_t can be parameterized by any neural network, which can depend on $\mathbf{u}_t$ and t in arbitrary ways. On the other hand, their disadvantage is that they necessitate using an ODE solver for sampling and calculating the density under the model, which can be slow compared to a single pass through a neural network. Additionally, unlike a neural network which has a fixed evaluation time, the evaluation time of an ODE solver may vary depending on the value of its input (i.e. its initial state).

2.6 Generative models without tractable densities

So far in this chapter I have focused on neural density estimators, that is, models whose density can be calculated efficiently. However, neural density estimation is not the only approach to generative modelling. Before I conclude the chapter, I will take a step back and briefly review generative models without tractable densities. I will discuss how such models relate to neural density estimators, how they differ from them, and what alternative capabilities they may offer.

2.6.1 Energy-based models

In section 2.2.4, we defined a neural density estimator to be a neural network that takes a vector $\mathbf{x}$ and returns a real number $f_{\boldsymbol{\phi}}(\mathbf{x})$ such that for any parameter setting $\boldsymbol{\phi}$:

$$\int_{\mathbb{R}^D} \exp(f_{\boldsymbol{\phi}}(\mathbf{x}))\,\mathrm{d}\mathbf{x} = 1. \tag{2.61}$$

The above property allows us to interpret $q_\phi(\mathbf{x}) = \exp(f_{\boldsymbol{\phi}}(\mathbf{x}))$ as a density function. In order to enforce this property, we had to restrict the architecture of the neural network, which as we saw can hurt the flexibility of the model.

We can relax this restriction by requiring only that the integral of $\exp(f_{\boldsymbol{\phi}}(\mathbf{x}))$ be finite. That is, for every parameter setting $\boldsymbol{\phi}$:

$$\int_{\mathbb{R}^D} \exp(f_{\boldsymbol{\phi}}(\mathbf{x}))\,\mathrm{d}\mathbf{x} = Z_{\boldsymbol{\phi}} < \infty. \tag{2.62}$$

If that's the case, we can still define a valid density function as follows:

$$q_{\boldsymbol{\phi}}(\mathbf{x}) = \frac{1}{Z_{\boldsymbol{\phi}}} \exp(f_{\boldsymbol{\phi}}(\mathbf{x})). \tag{2.63}$$

Models defined this way are called *energy-based models*. The quantity $-f_{\boldsymbol{\phi}}(\mathbf{x})$ is known as the *energy*, and $Z_{\boldsymbol{\phi}}$ is known as the *normalizing constant*. Under the above definitions, a neural density estimator is an energy-based model whose normalizing constant is always 1.

The main advantage of energy-based modelling is the increased flexibility in specifying $f_{\boldsymbol{\phi}}(\mathbf{x})$. However, evaluating the density under an energy-based model is intractable, since calculating the normalizing constant involves a high-dimensional integral. There are ways to estimate the normalizing constant, e.g. via *annealed importance sampling* (Salakhutdinov and Murray, 2008) or *model distillation* (Papamakarios and Murray, 2015), but they are approximate and often expensive.

In addition to their density being intractable, energy-based models typically don't provide a mechanism of generating samples. Instead, one must resort to approximate methods such as Markov-chain Monte Carlo (Murray, 2007; Neal, 1993) to generate samples. In contrast, some neural density estimators such *Masked Autoregressive Flow* or *Real NVP* are capable of generating exact independent samples directly.

Finally, the intractability of $q_{\boldsymbol{\phi}}(\mathbf{x})$ makes likelihood-based training of energy-based models less straightforward than of neural density estimators. In particular, the gradient of $\log q_{\boldsymbol{\phi}}(\mathbf{x})$ with respect to $\boldsymbol{\phi}$ is:

$$\frac{\partial}{\partial \boldsymbol{\phi}} \log q_{\boldsymbol{\phi}}(\mathbf{x}) = \frac{\partial}{\partial \boldsymbol{\phi}} f_{\boldsymbol{\phi}}(\mathbf{x}) - \frac{\partial}{\partial \boldsymbol{\phi}} \log Z_{\boldsymbol{\phi}}. \tag{2.64}$$

By substituting in the definition of the normalizing constant and by exchanging the order of differentiation and integration, we can write the gradient of $\log Z_{\boldsymbol{\phi}}$ as follows:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\phi}} \log Z_{\boldsymbol{\phi}} &= \frac{1}{Z_{\boldsymbol{\phi}}} \frac{\partial}{\partial \boldsymbol{\phi}} \int_{\mathbb{R}^D} \exp(f_{\boldsymbol{\phi}}(\mathbf{x}))\,\mathrm{d}\mathbf{x} && (2.65)\\
&= \int_{\mathbb{R}^D} \frac{1}{Z_{\boldsymbol{\phi}}} \exp(f_{\boldsymbol{\phi}}(\mathbf{x})) \frac{\partial}{\partial \boldsymbol{\phi}} f_{\boldsymbol{\phi}}(\mathbf{x})\,\mathrm{d}\mathbf{x} && (2.66)\\
&= \int_{\mathbb{R}^D} q_{\boldsymbol{\phi}}(\mathbf{x}) \frac{\partial}{\partial \boldsymbol{\phi}} f_{\boldsymbol{\phi}}(\mathbf{x})\,\mathrm{d}\mathbf{x} && (2.67)\\
&= \mathbb{E}_{q_{\boldsymbol{\phi}}(\mathbf{x})} \left(\frac{\partial}{\partial \boldsymbol{\phi}} f_{\boldsymbol{\phi}}(\mathbf{x}) \right). && (2.68)
\end{aligned}$$

The above expectation is intractable, hence direct gradient-based optimization of the average log likelihood is impractical. There are ways to either approximate the above expectation or avoid computing it entirely, such as *contrastive divergence* (Hinton, 2002), *score matching* (Hyvärinen, 2005), or *noise-contrastive estimation* (Gutmann and Hyvärinen, 2012). In contrast, for neural density estimators the above expectation is always zero by construction, which makes likelihood-based training by backpropagation readily applicable.

2.6.2 Latent-variable models and variational autoencoders

One way of increasing the expressivity of a simple generative model is by introducing *latent variables*. Latent variables are auxiliary variables $\mathbf{z}$ that are used to augment the data $\mathbf{x}$. For the purposes of this discussion we will assume that $\mathbf{z}$ is continuous (i.e. $\mathbf{z} \in \mathbb{R}^K$), but discrete latent variables are also possible.

Having decided on how many latent variables to introduce, one would typically define a joint density model $q_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z})$. This is often done by separately defining a prior model $q_{\boldsymbol{\phi}}(\mathbf{z})$ and a conditional model $q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z})$. Then, the density of $\mathbf{x}$ is obtained by:

$$q_{\boldsymbol{\phi}}(\mathbf{x}) = \int_{\mathbb{R}^K} q_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{z}) \,\mathrm{d}\mathbf{z} = \int_{\mathbb{R}^K} q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z}) q_{\boldsymbol{\phi}}(\mathbf{z}) \,\mathrm{d}\mathbf{z}. \tag{2.69}$$

The motivation for introducing latent variables and then integrating them out is that the marginal $q_{\boldsymbol{\phi}}(\mathbf{x})$ can be complex even if the prior $q_{\boldsymbol{\phi}}(\mathbf{z})$ and the conditional $q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z})$ are not. Hence, one can obtain a complex model of the data despite using simple modelling components. Another way to view $q_{\boldsymbol{\phi}}(\mathbf{x})$ is as a mixture of infinitely many components $q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z})$ indexed by $\mathbf{z}$, weighted by $q_{\boldsymbol{\phi}}(\mathbf{z})$. In fact, mixture models of finitely many components such as those discussed in section 2.2.1 can be thought of as latent-variable models with discrete latent variables.

Like energy-based models, the density of a latent-variable model is intractable to calculate since it involves a high-dimensional integral. If that integral is tractable, the model is essentially a neural density estimator and can be treated as such. So, for the purposes of this discussion, we'll assume that the integral is always intractable by definition.

Even though the model density is intractable, it's possible to lower-bound $\log q_{\boldsymbol{\phi}}(\mathbf{x})$ by introducing an auxiliary density model $r_{\boldsymbol{\psi}}(\mathbf{z} \,|\, \mathbf{x})$ with parameters $\boldsymbol{\psi}$ as follows:

$$\begin{aligned}
\log q_{\boldsymbol{\phi}}(\mathbf{x}) &= \log \int_{\mathbb{R}^K} q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z}) q_{\boldsymbol{\phi}}(\mathbf{z}) \,\mathrm{d}\mathbf{z} && (2.70)\\
&= \log \, \mathbb{E}_{r_{\boldsymbol{\psi}}(\mathbf{z} \,|\, \mathbf{x})} \left(\frac{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z}) q_{\boldsymbol{\phi}}(\mathbf{z})}{r_{\boldsymbol{\psi}}(\mathbf{z} \,|\, \mathbf{x})} \right) && (2.71)\\
&\geq \mathbb{E}_{r_{\boldsymbol{\psi}}(\mathbf{z} \,|\, \mathbf{x})} (\log q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z}) + \log q_{\boldsymbol{\phi}}(\mathbf{z}) - \log r_{\boldsymbol{\psi}}(\mathbf{z} \,|\, \mathbf{x})), && (2.72)
\end{aligned}$$

where we used *Jensen's inequality* to exchange the logarithm with the expectation. The above lower bound is often called *evidence lower bound* or *ELBO*. The ELBO is indeed a lower bound given any choice of conditional density $r_{\boldsymbol{\psi}}(\mathbf{z} \,|\, \mathbf{x})$, and it becomes equal to $\log q_{\boldsymbol{\phi}}(\mathbf{x})$ if:

$$r_{\boldsymbol{\psi}}(\mathbf{z} \,|\, \mathbf{x}) = q_{\boldsymbol{\phi}}(\mathbf{z} \,|\, \mathbf{x}) = \frac{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \mathbf{z}) q_{\boldsymbol{\phi}}(\mathbf{z})}{q_{\boldsymbol{\phi}}(\mathbf{x})}. \tag{2.73}$$

In practice, the ELBO can be used as a tractable objective to train the model. Maximizing the average ELBO over training data with respect to $\boldsymbol{\phi}$ is equivalent to maximizing a lower bound to the average log likelihood. Moreover, maximizing the average ELBO with respect to $\boldsymbol{\psi}$ makes the ELBO a tighter lower bound. Stochastic gradients of the average ELBO with respect to $\boldsymbol{\phi}$ and

$\boldsymbol{\psi}$ can be obtained by reparameterizing $r_{\boldsymbol{\psi}}(\mathbf{z} \mid \mathbf{x})$ with respect to its internal random variables (similarly to how we reparameterized MADE in section 2.3), and by estimating the expectation over the reparameterized $r_{\boldsymbol{\psi}}(\mathbf{z} \mid \mathbf{x})$ via Monte Carlo. A latent-variable model trained this way is known as a *variational autoencoder* (Kingma and Welling, 2014; Rezende et al., 2014). In the context of variational autoencoders, $r_{\boldsymbol{\psi}}(\mathbf{z} \mid \mathbf{x})$ is often referred to as the *encoder* and $q_{\boldsymbol{\phi}}(\mathbf{x} \mid \mathbf{z})$ as the *decoder*.

Apart from increasing the flexibility of the model, another motivation for introducing latent variables is *representation learning*. Consider for example a variational autoencoder that is used to model images of handwritten characters. Assume that the decoder $q_{\boldsymbol{\phi}}(\mathbf{x} \mid \mathbf{z})$ is deliberately chosen to factorize over the elements of $\mathbf{x}$, i.e. the pixels in an image. Clearly, a factorized distribution over pixels is unable to model global structure, which in our case corresponds to factors of variation such as the identity of a character, its shape, the style of handwriting, and so on. Therefore, information about global factors of variation such as the above must be encoded into the latent variables $\mathbf{z}$ if the model is to represent the data well. If the dimensionality of $\mathbf{z}$ is deliberately chosen to be smaller than that of $\mathbf{x}$ (the number of pixels), the latent variables correspond to a compressed encoding of the global factors of variation in an image. Given an image $\mathbf{x}$, plausible such encodings can be directly sampled from $r_{\boldsymbol{\psi}}(\mathbf{z} \mid \mathbf{x})$.

Finally, we can further encourage the latent variables to acquire semantic meaning by engineering the structure of the model appropriately. Examples include: hierarchically-structured latent variables that learn to represent datasets (Edwards and Storkey, 2017), exchangeable groups of latent variables that learn to represent objects in a scene (Eslami et al., 2016; Nash et al., 2017), and latent variables with a Markovian structure that learn to represent environment states (Buesing et al., 2018; Gregor et al., 2019).

2.6.3 Neural samplers and generative adversarial networks

A *neural sampler* is a neural network $f_{\boldsymbol{\phi}}$ with parameters $\boldsymbol{\phi}$ that takes random unstructured noise $\mathbf{u} \in \mathbb{R}^K$ and transforms it into data $\mathbf{x} \in \mathbb{R}^D$:

$$\mathbf{x} = f_{\boldsymbol{\phi}}(\mathbf{u}) \quad \text{where} \quad \mathbf{u} \sim \pi_u(\mathbf{u}). \tag{2.74}$$

In the above, $\pi_u(\mathbf{u})$ is a fixed source of noise (not necessarily a density model). Under this definition, normalizing flows with tractable sampling, such as Masked Autoregressive Flow or Real NVP, are also neural samplers. A typical variational autoencoder is also a neural sampler, in which case $\mathbf{u}$ are the internal random numbers needed to sample $\mathbf{z}$ from the prior and $\mathbf{x}$ from the decoder.

Typically, the main purpose of a neural sampler is to be used as a data generator and not as a density estimator. Hence, a neural sampler doesn't need to have the restrictions of a normalizing flow: $f_{\boldsymbol{\phi}}$ doesn't have to be bijective and K (the dimensionality of the noise) doesn't have to match D (the dimensionality of the data). As a result, a neural sampler offers increased flexibility in specifying the neural network $f_{\boldsymbol{\phi}}$ compared to a normalizing flow.

A consequence of the freedom of specifying $f_{\boldsymbol{\phi}}$ and K is that a neural sampler may no longer correspond to a valid density model of the data. For example, if $f_{\boldsymbol{\phi}}$ is not injective, a whole subset of $\mathbb{R}^K$ of non-zero measure under $\pi_u(\mathbf{u})$ can be mapped to a single point in $\mathbb{R}^D$, where the model density will be infinite. Moreover, if $K < D$, the neural sampler can only generate data on a K-dimensional manifold embedded in $\mathbb{R}^D$. Hence, the model density will be zero almost everywhere, except for a manifold of zero volume where the model density will be infinite.

Training a neural sampler is typically done by generating a set of samples from the model, selecting a random subset of the training data, and calculating a measure of discrepancy between

the two sets. If that discrepancy measure is differentiable, its gradient with respect to the model parameters $\boldsymbol{\phi}$ can be obtained by backpropagation. Then, the model can be trained by stochastic-gradient descent, with stochastic gradients obtained by repeating the above procedure.

One way to obtain a differentiable discrepancy measure is via an additional neural network, known as the *discriminator*, whose task is to discriminate generated samples from training data. The classification performance of the discriminator (as measured by e.g. cross entropy) can be used as a differentiable discrepancy measure. In practice, the discriminator is trained at the same time as the neural sampler. Neural samplers trained this way are known as *generative adversarial networks* (Goodfellow et al., 2014). Other possible discrepancy measures between training data and generated samples include the maximum mean discrepancy (Dziugaite et al., 2015), or the Wasserstein distance (Arjovsky et al., 2017).

Generative adversarial networks often excel at data-generation performance. When trained as image models, they are able to generate high-resolution images that are visually indistinguishable from real images (Brock et al., 2019; Karras et al., 2018a,b). However, their good performance as data generators doesn't necessarily imply good performance in terms of average log likelihood. Grover et al. (2018) and Danihelka et al. (2017) have observed that a Real NVP trained as a generative adversarial network can generate realistic-looking samples, but it may perform poorly as a density estimator.

2.7 Summary and conclusions

The probability density of a generative process at a location $\mathbf{x}$ is how often the process generates data near $\mathbf{x}$ per unit volume. Density estimation is the task of modelling the density of a process given training data generated by the process. In this chapter, I discussed how to perform density estimation with neural networks, and why this may be a good idea.

Instead of densities, we could directly estimate other statistical quantities associated with the process, such as expectations of various functions under the process, or we could model the way data is generated by the process. Why estimate densities then? In section 2.1.1, I identified the following reasons as to why estimating densities can be useful:

(i) Bayesian inference is a calculus over densities.

(ii) Accurate densities imply good data compression.

(iii) The density of a model is a principled training and evaluation objective.

(iv) Density models are useful components in other algorithms.

That said, I would argue that in practice we should first identify what task we are interested in, and then use the tool that is more appropriate for the task. If the task calls for a density model, then a density model should be used. Otherwise, a different solution may be more appropriate.

Estimating densities is hard, and it becomes dramatically harder as the dimensionality of the data increases. As I discussed in section 2.1.2, the reason is the curse of dimensionality: a dataset becomes sparser in higher dimensions, and naive density estimation doesn't scale. In section 2.2, I argued that standard density estimation methods based on histograms or smoothing kernels are particularly susceptible to the curse of dimensionality.

Is high-dimensional density estimation hopeless? In section 2.1.2 I argued that it's not, as long as we are prepared to make assumptions about the densities we want to estimate, and encode these assumptions into our models. I identified the following assumptions that are often appropriate to make for real-world densities:

(i) They vary smoothly.

(ii) Their intrinsic dimensionality is smaller than that of the data.

(iii) They have symmetries.

(iv) They involve variables that are loosely dependent.

Neural density estimators scale to high dimensions partly because their architecture can encode the above assumptions. In contrast, it's not clear how these assumptions can be encoded into models such as parametric mixtures, histograms, or smoothing kernels.

The main contribution of this chapter is *Masked Autoregressive Flow*, a normalizing flow that consists of a stack of reparameterized autoregressive models. In section 2.4, I discussed the impact Masked Autoregressive Flow has had since its publication, how it contributed to further research, and what its main limitations have been.

Since the publication of Masked Autoregressive Flow in December 2017, the field of normalizing flows has experienced considerable research interest. In section 2.5, I reviewed advances in the field since the publication of the paper, and I examined their individual strengths and weaknesses. Even though normalizing flows have improved significantly, no model yet exists that has all the following characteristics:

(i) The density under the model can be exactly evaluated in one neural-network pass (like Masked Autoregressive Flow).

(ii) Exact independent samples can be generated in one neural-network pass (like Inverse Autoregressive Flow).

(iii) The modelling performance is on par with the best autoregressive models.

Of all the models that satisfy both (i) and (ii), the best-performing ones are Real NVP and its successor Glow. However, as we saw in section 2.3, the coupling layers used by Real NVP and Glow are not as expressive as the autoregressive layers used by MAF and IAF. On the other hand, satisfying (iii) often requires sacrificing either (i) or (ii). Nonetheless, there is a lot of research activity currently in normalizing flows, and several new ideas such as Neural ODEs and FFJORD are beginning to emerge, so it's reasonable to expect further advances in the near future.

Neural density estimation is only one approach to generative modelling, with variational autoencoders and generative adversarial networks being the main alternatives when exact density evaluations are not required. As I discussed in section 2.6, the advantage of variational autoencoders is their ability to learn structured representations, whereas the strength of generative adversarial networks is their performance in generating realistic data. As I argued earlier, choosing among a neural density estimator, a variational autoencoder and a generative adversarial network should primarily depend on the task we are interested in.

That said, normalizing flows, variational autoencoders and generative adversarial networks are all essentially the same model: a neural network that takes unstructured noise and turns it into data. The difference is in the requirements the different models must satisfy (a normalizing flow must be invertible), in the capabilities they offer (a normalizing flow provides explicit densities in addition to sampling), and in the way they are trained. The fact that generative adversarial networks can take white noise and turn it into convincingly realistic images should be seen as proof that (a) this task can be achieved by a smooth function, and (b) such a function is learnable from data. Since this task is demonstrably possible, it should serve as a performance target for neural density estimators too.

Part II

Likelihood-free inference

Chapter 3

Fast ϵ-free Inference of Simulator Models with Bayesian Conditional Density Estimation

This chapter is about likelihood-free inference, i.e. Bayesian inference when the likelihood is intractable, and how neural density estimation can be used to address this challenge. We start by explaining what likelihood-free inference is, in what situations the likelihood may be intractable, and why it is important to address this challenge (section 3.1). We then review *approximate Bayesian computation*, a family of methods for likelihood-free inference based on simulations (section 3.2). The main contribution of this chapter is the paper *Fast ϵ-free Inference of Simulator Models with Bayesian Conditional Density Estimation*, which introduces a new method for likelihood-free inference based on neural density estimation (section 3.3). We conclude the chapter with a discussion of the contributions, impact and limitations of the paper, and a comparison of the proposed method with previous and following work (section 3.4).

3.1 Likelihood-free inference: what and why?

Suppose we have a stationary process that generates data and is controlled by parameters $\boldsymbol{\theta}$. Every time the process is run forward, it stochastically generates a datapoint $\mathbf{x}$ whose distribution depends on $\boldsymbol{\theta}$. We will assume that $\mathbf{x}$ and $\boldsymbol{\theta}$ are continuous vectors, but many of the techniques discussed in this thesis can be adapted for discrete $\mathbf{x}$ and/or $\boldsymbol{\theta}$. We will further assume that for every setting of $\boldsymbol{\theta}$, the corresponding distribution over $\mathbf{x}$ admits a finite density everywhere; in other words, the process defines a conditional density function $p(\mathbf{x} \,|\, \boldsymbol{\theta})$.

Given a datapoint $\mathbf{x}_o$ known to have been generated by the process, we are interested in inferring plausible parameter settings that could have generated $\mathbf{x}_o$. In particular, given prior beliefs over parameters described by a density $p(\boldsymbol{\theta})$, we are interested in computing the posterior density $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ obtained by *Bayes' rule*:

$$p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) = \frac{p(\mathbf{x}_o \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{Z(\mathbf{x}_o)} \quad \text{where} \quad Z(\mathbf{x}_o) = \int p(\mathbf{x}_o \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}')\, \mathrm{d}\boldsymbol{\theta}'. \tag{3.1}$$

We assume that the normalizing constant $Z(\mathbf{x}_o)$ is finite for every $\mathbf{x}_o$, so that the posterior density is always well-defined.

3.1.1 Density models and simulator models

The choice of inference algorithm depends primarily on how the generative process is modelled. In this section, I discuss and compare two types of models: density models and simulator models.

A *density model*, also known as an *explicit model*, describes the conditional density function of the process. Given values for $\mathbf{x}$ and $\boldsymbol{\theta}$, a density model returns the value of the conditional density $p(\mathbf{x} \mid \boldsymbol{\theta})$ (or an approximation of it if the model is not exact). With a density model, we can easily evaluate the posterior density $p(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$ up to a normalizing constant using Bayes' rule (equation 3.1). Even though the normalizing constant $Z(\mathbf{x}_o)$ is typically intractable, we can sample from the posterior using e.g. Markov-chain Monte Carlo (Murray, 2007; Neal, 1993), or approximate the posterior with a more convenient distribution using e.g. variational inference (Blei et al., 2017; Kucukelbir et al., 2015; Ranganath et al., 2014) or knowledge distillation (Papamakarios, 2015; Papamakarios and Murray, 2015). We refer to such methods as *likelihood-based inference methods*, as they explicitly evaluate the likelihood $p(\mathbf{x}_o \mid \boldsymbol{\theta})$.

On the other hand, a *simulator model*, also known as an *implicit model*, describes how the process generates data. For any parameter setting $\boldsymbol{\theta}$, a simulator model can be run forward to generate exact independent samples from $p(\mathbf{x} \mid \boldsymbol{\theta})$ (or approximate independent samples if the model is not perfect). Unlike a density model, we can't typically evaluate the density under a simulator model. In order to perform inference in a simulator model we need methods that make use of simulations from the model but don't require density evaluations. We refer to such methods as *likelihood-free inference methods*.

In general, likelihood-free methods are less efficient than likelihood-based methods. As we will see in sections 3.2 and 3.3, likelihood-free methods can require a large number of simulations from the model to produce accurate results, even for fairly simple models. One of the main topics of this thesis, and of research in likelihood-free inference in general, is how to reduce the required number of simulations without sacrificing inference quality.

Since likelihood-free methods are less efficient than likelihood-based methods, why should we bother with simulator models at all, and not just require all models to be density models instead? The answer is that simulators can often be more natural and more interpretable modelling tools than density functions. A simulator model is a direct specification of how a process of interest generates data, which can be closer to our understanding of the process as a working mechanism. In contrast, a density function is a mathematical abstraction that is often hard to interpret, and as a result it can be an unintuitive description of real-world phenomena.

Simulator models are particularly well-suited for scientific applications. In high-energy physics for instance, simulators are used to model collisions in particle accelerators and the interaction of produced particles with particle detectors (Agostinelli et al., 2003; Sjöstrand et al., 2008). Inference in such models can be used to infer parameters of physical theories and potentially discover new physics (Brehmer et al., 2018a,b). Other scientific applications of simulator models and likelihood-free inference methods include astronomy and cosmology (Alsing et al., 2018a, 2019; Schafer and Freeman, 2012), systems biology (Wilkinson, 2011), evolution and ecology (Beaumont, 2010; Pritchard et al., 1999; Ratmann et al., 2007; Wood, 2010), population generics (Beaumont et al., 2002), and computational neuroscience (Lueckmann et al., 2017; Markram et al., 2015; Pospischil et al., 2008; Sterratt et al., 2011)

In addition to their scientific applications, simulator models can be useful in engineering and artificial intelligence. For instance, one can build an image model using a computer-graphics engine: the engine takes a high-level description of a scene (which corresponds to the model parameters $\boldsymbol{\theta}$) and renders an image of that scene (which corresponds to the generated data $\mathbf{x}$). Computer vision and scene understanding can be cast as inference of scene parameters given an observed image (Kulkarni et al., 2015; Mansinghka et al., 2013; Romaszko et al., 2017).

Other examples of applications of simulator models in artificial intelligence include perceiving physical properties of objects using a physics engine (Wu et al., 2015), and inferring the goals of autonomous agents using probabilistic programs (Cusumano-Towner et al., 2017).

3.1.2 When is the likelihood intractable?

In the previous section, I argued that simulator models are useful modelling tools. Nonetheless, this doesn't necessitate the use of likelihood-free inference methods: given a simulator model, we could try to calculate its likelihood based on the simulator's internal workings, and then use likelihood-based inference methods instead. In this section, I will discuss certain cases in which it may be too expensive or even impossible to calculate the likelihood of a simulator, and thus likelihood-free methods may be preferable or necessary.

Integrating over the simulator's latent state

Typically, the stochasticity of a simulator model comes from internal generation of random numbers, which form the simulator's latent state. Consider for example a simulator whose latent state consists of variables $\mathbf{z}_1, \ldots, \mathbf{z}_K$, generated from distributions $p_{\boldsymbol{\psi}_1}, \ldots, p_{\boldsymbol{\psi}_K}$ whose parameters $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_K$ can be arbitrary functions of all variables preceding them. For simplicity, assume that all variables are continuous, and that K is fixed (and not random). Such a simulator can be expressed as the following sequence of statements:

$$\begin{aligned}
\boldsymbol{\psi}_1 &= f_1(\boldsymbol{\theta}) && (3.2)\\
\mathbf{z}_1 &\sim p_{\boldsymbol{\psi}_1} && (3.3)\\
&\ldots \\
\boldsymbol{\psi}_K &= f_K(\mathbf{z}_{1:K-1}, \boldsymbol{\theta}) && (3.4)\\
\mathbf{z}_K &\sim p_{\boldsymbol{\psi}_K} && (3.5)\\
\boldsymbol{\phi} &= g(\mathbf{z}_{1:K}, \boldsymbol{\theta}) && (3.6)\\
\mathbf{x} &\sim p_{\boldsymbol{\phi}}. && (3.7)
\end{aligned}$$

If the distributions $p_{\boldsymbol{\psi}_1}, \ldots, p_{\boldsymbol{\psi}_K}$ and $p_{\boldsymbol{\phi}}$ admit tractable densities, the joint density of data $\mathbf{x}$ and latent state $\mathbf{z}_{1:K}$ is also tractable, and can be calculated by:

$$p(\mathbf{x}, \mathbf{z}_{1:K} \mid \boldsymbol{\theta}) = p_{\boldsymbol{\phi}}(\mathbf{x}) \prod_k p_{\boldsymbol{\psi}_k}(\mathbf{z}_k). \tag{3.8}$$

Given observed data $\mathbf{x}_o$, the likelihood of the simulator is:

$$p(\mathbf{x}_o \mid \boldsymbol{\theta}) = \int p(\mathbf{x}_o, \mathbf{z}_{1:K} \mid \boldsymbol{\theta}) \, \mathrm{d}\mathbf{z}_{1:K}. \tag{3.9}$$

Since the likelihood of the simulator involves integration over the latent state, it is generally intractable.

Even when integration over the latent state makes the likelihood intractable, the joint likelihood $p(\mathbf{x}_o, \mathbf{z}_{1:K} \mid \boldsymbol{\theta})$ may still be tractable (as in the above example). We could then sample from the joint posterior $p(\mathbf{z}_{1:K}, \boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$ using e.g. MCMC, and simply discard the latent-state samples. However, this may be infeasible when the latent state is large. Consider for example a simulator of a physical process involving Brownian motion of a large set of particles. Inferring the latent state of the simulator would entail sampling from the space of all possible particle trajectories that are consistent with the data. If both the observed data $\mathbf{x}_o$ and the parameters of interest $\boldsymbol{\theta}$ correspond to macroscopic variables associated with the process, likelihood-free inference methods that don't involve inferring the microscopic state of the process may be preferable.

Deterministic transformations

Sometimes, the likelihood of a simulator model is intractable because the output of the simulator is a deterministic transformation of the simulator's latent state. Consider for example the following simulator model:

$$\mathbf{z} \sim p(\mathbf{z} \,|\, \boldsymbol{\theta}) \tag{3.10}$$
$$\mathbf{x} = f(\mathbf{z}, \boldsymbol{\theta}), \tag{3.11}$$

where $\mathbf{z}$ is continuous and the density $p(\mathbf{z} \,|\, \boldsymbol{\theta})$ is tractable. The joint distribution of $\mathbf{x}$ and $\mathbf{z}$ does not admit a proper density, but can be expressed using a delta distribution as follows:

$$p(\mathbf{x}, \mathbf{z} \,|\, \boldsymbol{\theta}) = \delta(\mathbf{x} - f(\mathbf{z}, \boldsymbol{\theta}))\, p(\mathbf{z} \,|\, \boldsymbol{\theta}). \tag{3.12}$$

The likelihood of the simulator is:

$$p(\mathbf{x}_o \,|\, \boldsymbol{\theta}) = \int \delta(\mathbf{x}_o - f(\mathbf{z}, \boldsymbol{\theta}))\, p(\mathbf{z} \,|\, \boldsymbol{\theta})\, \mathrm{d}\mathbf{z}. \tag{3.13}$$

Calculating the above integral requires a change of variables from $\mathbf{z}$ to $\mathbf{x}$ which is not tractable in general. Moreover, the distribution $p(\mathbf{x} \,|\, \boldsymbol{\theta})$ may not admit a proper density even if $p(\mathbf{z} \,|\, \boldsymbol{\theta})$ does.

Sampling from the joint posterior of $\mathbf{z}$ and $\boldsymbol{\theta}$ in order to avoid calculating the likelihood also poses difficulties. Given observed data $\mathbf{x}_o$, the joint posterior $p(\mathbf{z}, \boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ is constrained on a manifold in the $(\mathbf{z}, \boldsymbol{\theta})$-space defined by $\mathbf{x}_o = f(\mathbf{z}, \boldsymbol{\theta})$. One approach to sampling from this posterior is to replace the joint distribution $p(\mathbf{x}, \mathbf{z} \,|\, \boldsymbol{\theta})$ with the following approximation:

$$p_\epsilon(\mathbf{x}, \mathbf{z} \,|\, \boldsymbol{\theta}) = \mathcal{N}\big(\mathbf{x} \,|\, f(\mathbf{z}, \boldsymbol{\theta}), \epsilon^2 \mathbf{I}\big)\, p(\mathbf{z} \,|\, \boldsymbol{\theta}), \tag{3.14}$$

where ϵ is a small positive constant. This is equivalent to replacing the original simulator model with the following one:

$$\mathbf{z} \sim p(\mathbf{z} \,|\, \boldsymbol{\theta}) \tag{3.15}$$
$$\mathbf{z}' \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{3.16}$$
$$\mathbf{x} = f(\mathbf{z}, \boldsymbol{\theta}) + \epsilon \mathbf{z}'. \tag{3.17}$$

An issue with this approach is that for small ϵ methods such as MCMC may struggle, whereas for large ϵ the posterior no longer corresponds to the original model. MCMC methods on constrained manifolds exist and may be used instead, but make assumptions about the manifold which may not hold for a simulator model in general (Brubaker et al., 2012; Graham and Storkey, 2017).

Inaccessible internal workings

Even if it is theoretically possible to use a likelihood-based method with a simulator model, this may be difficult in practice if the simulator is a black box whose internal workings we don't have access to. For example, the simulator may be provided as an executable program, or it may be written in a low-level language (e.g. for running efficiency), or it may be an external library routine. I would argue that a general-purpose inference engine ought to have inference algorithms that support such cases.

Finally, the "simulator" may not be a model written in computer code, but an actual process in the real world. For example, the "simulator" could be an experiment in a lab, asking participants to perform a task, or a measurement of a physical phenomenon. If a model doesn't exist, likelihood-free inference methods could be used directly in the real world.

3.2 Approximate Bayesian computation

In this section, I will discuss a family of likelihood-free inference methods broadly known as *approximate Bayesian computation* or *ABC*. In general, ABC methods draw approximate samples from the parameter posterior by repeatedly simulating the model and checking whether simulated data match the observed data. I will discuss three types of ABC methods: *rejection ABC*, *Markov-chain Monte Carlo ABC*, and *sequential Monte Carlo ABC*.

3.2.1 Rejection ABC

Let $B_\epsilon(\mathbf{x}_o)$ be a neighbourhood of $\mathbf{x}_o$, defined as the set of datapoints whose distance from $\mathbf{x}_o$ is no more than ϵ:

$$B_\epsilon(\mathbf{x}_o) = \{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon\}. \tag{3.18}$$

In the above, $\|\cdot\|$ can be the Euclidean norm, or any other norm in $\mathbb{R}^D$. For a small ϵ, we can approximate the likelihood $p(\mathbf{x}_o \,|\, \boldsymbol{\theta})$ by the average conditional density inside $B_\epsilon(\mathbf{x}_o)$:

$$p(\mathbf{x}_o \,|\, \boldsymbol{\theta}) \approx \frac{\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta})}{|B_\epsilon(\mathbf{x}_o)|}, \tag{3.19}$$

where $|B_\epsilon(\mathbf{x}_o)|$ is the volume of $B_\epsilon(\mathbf{x}_o)$. Substituting the above likelihood approximation into Bayes' rule (equation 3.1), we obtain the following approximation to the posterior:

$$p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) \approx \frac{\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int \Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}')\, \mathrm{d}\boldsymbol{\theta}'} = p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon). \tag{3.20}$$

The approximate posterior $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$ can be thought of as the exact posterior under an alternative observation, namely that generated data $\mathbf{x}$ falls inside the neighbourhood $B_\epsilon(\mathbf{x}_o)$. As ϵ approaches zero, $B_\epsilon(\mathbf{x}_o)$ becomes infinitesimally small, and the approximate posterior approaches the exact posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ (for a formal proof under suitable regularity conditions, see e.g. Prangle, 2017, supplementary material, theorem 1). Conversely, as ϵ approaches infinity, $B_\epsilon(\mathbf{x}_o)$ covers the whole space, and the approximate posterior approaches the prior $p(\boldsymbol{\theta})$.

Rejection ABC (Pritchard et al., 1999) is a rejection-sampling method for obtaining independent samples from the approximate posterior $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$. It works by first sampling parameters from the prior $p(\boldsymbol{\theta})$, then simulating the model using the sampled parameters, and only accepting the sample if the simulated data is no further than a distance ϵ away from $\mathbf{x}_o$. The parameter ϵ controls the tradeoff between approximation quality and computational efficiency: as ϵ becomes smaller, the accepted samples follow more closely the exact posterior, but the algorithm accepts less often. Rejection ABC is shown in algorithm 3.1.

Algorithm 3.1: Rejection ABC

repeat
  Sample $\boldsymbol{\theta} \sim p(\boldsymbol{\theta})$
  Simulate $\mathbf{x} \sim p(\mathbf{x} \,|\, \boldsymbol{\theta})$
until $\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon$
return $\boldsymbol{\theta}$

An issue with rejection ABC is that it can become prohibitively expensive for small ϵ, especially when the data is high-dimensional. As an illustration, consider the case where $\|\cdot\|$ is the maximum

norm, and hence the acceptance region $B_\epsilon(\mathbf{x}_o)$ is a cube of side 2ϵ centred at $\mathbf{x}_o$. For small ϵ, the acceptance probability can be approximated as follows:

$$\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta}) \approx p(\mathbf{x}_o \,|\, \boldsymbol{\theta})\, |B_\epsilon(\mathbf{x}_o)| = p(\mathbf{x}_o \,|\, \boldsymbol{\theta})\, (2\epsilon)^D, \tag{3.21}$$

where D is the dimensionality of $\mathbf{x}$. As $\epsilon \to 0$, the acceptance probability approaches zero at a rate of $\mathcal{O}(\epsilon^D)$. Moreover, as D grows large, the acceptance probability approaches zero for any $\epsilon < 1/2$. To put that into perspective, if $p(\mathbf{x}_o \,|\, \boldsymbol{\theta}) = 1$, $\epsilon = 0.05$ and $D = 15$, we would need on average about a thousand trillion simulations for each accepted sample!

In practice, in order to maintain the acceptance probability at manageable levels, we typically transform the data into a small number of *features*, also known as *summary statistics*, in order to reduce the dimensionality D. If the summary statistics are sufficient for inferring $\boldsymbol{\theta}$, then turning data into summary statistics incurs no loss of information about $\boldsymbol{\theta}$ (for a formal definition of sufficiency, see e.g. Wasserman, 2010, definition 9.32). However, it can be hard to find sufficient summary statistics, and so in practice summary statistics are often chosen based on domain knowledge about the inference task. A large section of the ABC literature is devoted to constructing good summary statistics (see e.g. Blum et al., 2013; Charnock et al., 2018; Fearnhead and Prangle, 2012; Prangle et al., 2014, and references therein).

So far I've described rejection ABC as a rejection-sampling algorithm for the approximate posterior obtained by replacing $p(\mathbf{x}_o \,|\, \boldsymbol{\theta})$ by the average conditional density near $\mathbf{x}_o$. An alternative way to view rejection ABC is as a kernel density estimate of the joint density $p(\boldsymbol{\theta}, \mathbf{x})$ that has been conditioned on $\mathbf{x}_o$. Let $\{(\boldsymbol{\theta}_1, \mathbf{x}_1), \ldots, (\boldsymbol{\theta}_N, \mathbf{x}_N)\}$ be a set of joint samples from $p(\boldsymbol{\theta}, \mathbf{x})$, obtained as follows:

$$\boldsymbol{\theta}_n \sim p(\boldsymbol{\theta}) \quad \text{and} \quad \mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}_n). \tag{3.22}$$

Let $k_\epsilon(\cdot)$ be a smoothing kernel, and let $\delta(\cdot)$ be the delta distribution. We form the following approximation to the joint density, which is a kernel density estimate in $\mathbf{x}$ and an empirical distribution in $\boldsymbol{\theta}$:

$$\hat{p}(\boldsymbol{\theta}, \mathbf{x}) = \frac{1}{N} \sum_n k_\epsilon(\mathbf{x} - \mathbf{x}_n) \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_n). \tag{3.23}$$

By conditioning on $\mathbf{x}_o$ and normalizing, we obtain an approximate posterior in the form of a weighted empirical distribution:

$$\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) = \frac{\hat{p}(\boldsymbol{\theta}, \mathbf{x}_o)}{\int \hat{p}(\boldsymbol{\theta}, \mathbf{x}_o)\, \mathrm{d}\boldsymbol{\theta}} = \frac{\sum_n k_\epsilon(\mathbf{x}_o - \mathbf{x}_n) \delta(\boldsymbol{\theta} - \boldsymbol{\theta}_n)}{\sum_n k_\epsilon(\mathbf{x}_o - \mathbf{x}_n)}. \tag{3.24}$$

If we use the following uniform smoothing kernel:

$$k_\epsilon(\mathbf{u}) \propto I(\|\mathbf{u}\| \leq \epsilon) = \begin{cases} 1 & \|\mathbf{u}\| \leq \epsilon \\ 0 & \text{otherwise,} \end{cases} \tag{3.25}$$

then $\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ is simply the empirical distribution of samples returned by rejection ABC. The above can be seen as a way to generalize rejection ABC to *smooth-rejection ABC*, where we weight each prior sample $\boldsymbol{\theta}_n$ by a smoothing kernel $k_\epsilon(\mathbf{x}_o - \mathbf{x}_n)$ instead of accepting or rejecting it (Beaumont et al., 2002).

Additionally, we can interpret $\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ as the empirical distribution of the exact posterior of a different model, as given by importance sampling. Consider an alternative simulator model, obtained by adding noise $\mathbf{u} \sim k_\epsilon(\mathbf{u})$ to the output $\mathbf{x}$ of the original simulator:

$$\mathbf{x} \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}) \tag{3.26}$$

$$\mathbf{u} \sim k_\epsilon(\mathbf{u}) \tag{3.27}$$

$$\mathbf{x}' = \mathbf{x} + \mathbf{u}. \tag{3.28}$$

Suppose we observed $\mathbf{x}' = \mathbf{x}_o$, and want to perform inference on $\boldsymbol{\theta}$. The posterior $p(\boldsymbol{\theta}, \mathbf{x} \,|\, \mathbf{x}' = \mathbf{x}_o)$ can be written as:

$$p(\boldsymbol{\theta}, \mathbf{x} \,|\, \mathbf{x}' = \mathbf{x}_o) \propto k_\epsilon(\mathbf{x}_o - \mathbf{x})\, p(\boldsymbol{\theta}, \mathbf{x}). \tag{3.29}$$

We will form a weighted empirical distribution of the above posterior using importance sampling, with $p(\boldsymbol{\theta}, \mathbf{x})$ as the proposal. First, we sample $\{(\boldsymbol{\theta}_1, \mathbf{x}_1), \ldots, (\boldsymbol{\theta}_N, \mathbf{x}_N)\}$ from $p(\boldsymbol{\theta}, \mathbf{x})$, and then we form the following importance-weighted empirical distribution:

$$\hat{p}(\boldsymbol{\theta}, \mathbf{x} \,|\, \mathbf{x}' = \mathbf{x}_o) = \frac{\sum_n k_\epsilon(\mathbf{x}_o - \mathbf{x}_n)\delta(\boldsymbol{\theta} - \boldsymbol{\theta}_n)\delta(\mathbf{x} - \mathbf{x}_n)}{\sum_n k_\epsilon(\mathbf{x}_o - \mathbf{x}_n)}. \tag{3.30}$$

By marginalizing out $\mathbf{x}$ (i.e. by discarding samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$), we obtain the weighted empirical distribution $\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ as given by equation (3.24). This shows that smooth-rejection ABC, and by extension rejection ABC, can be understood as sampling from the exact posterior of an alternative model. In this interpretation, ϵ controls the extent to which the alternative model is different to the original one. This interpretation of rejection ABC as sampling from a different model was proposed by Wilkinson (2013) using rejection sampling; here I provide an alternative perspective using importance sampling instead.

3.2.2 Markov-chain Monte Carlo ABC

As we saw in the previous section, rejection ABC proposes parameters from the prior $p(\boldsymbol{\theta})$, and only accepts parameters that are likely under the approximate posterior $p(\boldsymbol{\theta} \mid \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$. If the approximate posterior is significantly narrower than the prior, as is often the case in practice, a large fraction of proposed parameters will be rejected.

An alternative approach that can lead to fewer rejections is to propose parameters using Markov-chain Monte Carlo in the form of *Metropolis–Hastings*. Instead of proposing parameters independently from the prior, Metropolis–Hastings proposes new parameters by e.g. perturbing previously accepted parameters. If the perturbation is not too large, the proposed parameters are likely to be accepted too, which can lead to fewer rejections compared to proposing from the prior.

A likelihood-based Metropolis–Hastings algorithm that targets $p(\boldsymbol{\theta} \mid \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$ is shown in algorithm 3.2. The algorithm uses a proposal distribution $q(\boldsymbol{\theta}' \,|\, \boldsymbol{\theta})$ from which it proposes new parameters $\boldsymbol{\theta}'$ based on previously accepted parameters $\boldsymbol{\theta}$. Then, the following quantity is calculated, known as the *acceptance ratio*:

$$\alpha = \frac{\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}')\, q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}')}{\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, q(\boldsymbol{\theta}' \,|\, \boldsymbol{\theta})}. \tag{3.31}$$

Finally, the algorithm outputs the proposed parameters $\boldsymbol{\theta}'$ with probability $\min(1, \alpha)$, otherwise it outputs the previous parameters $\boldsymbol{\theta}$. It can be shown that this procedure leaves the approximate posterior invariant; that is, if $\boldsymbol{\theta}$ is a sample from $p(\boldsymbol{\theta} \mid \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$, then the algorithm's output is also a sample from $p(\boldsymbol{\theta} \mid \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$ (for a proof, see Bishop, 2006, section 11.2.2).

In the likelihood-free setting, we cannot evaluate the approximate likelihood $\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta})$, but we can estimate it as the fraction of simulated data that are at most distance ϵ away from the observed data $\mathbf{x}_o$:

$$\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta}) \approx \frac{1}{N} \sum_n I(\|\mathbf{x}_n - \mathbf{x}_o\| \leq \epsilon) \quad \text{where} \quad \mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}). \tag{3.32}$$

The above estimate is unbiased for any number of simulations N, that is, its expectation is equal to $\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta})$. It can be shown that Metropolis–Hastings leaves the target distribution invariant even if we use an unbiased estimate of the target distribution instead of its exact value

Algorithm 3.2: Likelihood-based Metropolis–Hastings

Propose $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$

Calculate acceptance ratio $\alpha = \frac{\Pr\left(\|\mathbf{x}-\mathbf{x}_o\| \leq \epsilon \mid \boldsymbol{\theta}'\right) p\left(\boldsymbol{\theta}'\right) q\left(\boldsymbol{\theta} \mid \boldsymbol{\theta}'\right)}{\Pr(\|\mathbf{x}-\mathbf{x}_o\| \leq \epsilon \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})\, q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})}$

Sample $u \sim \mathcal{U}(0, 1)$

if $u \leq \alpha$ **then**

  return $\boldsymbol{\theta}'$

else

  return $\boldsymbol{\theta}$

in the acceptance ratio, as long as the estimate itself is made part of the Markov chain (for a proof, see Andrieu and Roberts, 2009). This means that we can use the above unbiased estimate of the approximate likelihood in the acceptance ratio and obtain a valid likelihood-free Metropolis–Hastings algorithm. This algorithm, which I refer to as *pseudo-marginal Metropolis–Hastings*, is shown in algorithm 3.3.

Algorithm 3.3: Pseudo-marginal Metropolis–Hastings

Propose $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})$

for $n = 1 : N$ **do**

  Simulate $\mathbf{x}_n \sim p(\mathbf{x} \mid \boldsymbol{\theta}')$

Calculate approximate-likelihood estimate $L' = \frac{1}{N} \sum_n I(\|\mathbf{x}_n - \mathbf{x}_o\| \leq \epsilon)$

Calculate acceptance ratio $\alpha = \frac{L'\, p\left(\boldsymbol{\theta}'\right) q\left(\boldsymbol{\theta} \mid \boldsymbol{\theta}'\right)}{L\, p(\boldsymbol{\theta})\, q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})}$

Sample $u \sim \mathcal{U}(0, 1)$

if $u \leq \alpha$ **then**

  return $(\boldsymbol{\theta}', L')$

else

  return $(\boldsymbol{\theta}, L)$

The estimate in equation (3.32) is unbiased for any number of simulations N, hence pseudo-marginal Metropolis–Hastings is valid for any N. For larger N, the approximate-likelihood estimate has lower variance, but the runtime of the algorithm increases as more simulations are required per step. Overall, as discussed by Bornn et al. (2017), the efficiency of the algorithm doesn't necessarily increase with N, and so $N = 1$ is usually good enough. In the case of $N = 1$, i.e. where only one datapoint $\mathbf{x}$ is simulated from $p(\mathbf{x} \mid \boldsymbol{\theta}')$ per step, and assuming $\boldsymbol{\theta}$ is a previously accepted sample, the acceptance ratio simplifies into the following:

$$\alpha = \begin{cases} \frac{p\left(\boldsymbol{\theta}'\right) q\left(\boldsymbol{\theta} \mid \boldsymbol{\theta}'\right)}{p(\boldsymbol{\theta})\, q(\boldsymbol{\theta}' \mid \boldsymbol{\theta})} & \text{if } \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon, \\ 0 & \text{otherwise.} \end{cases} \tag{3.33}$$

For $N = 1$, pseudo-marginal Metropolis–Hastings can be simplified into algorithm 3.4, which is known as *Markov-chain Monte Carlo ABC*. MCMC-ABC was proposed by Marjoram et al. (2003) prior to the introduction of pseudo-marginal MCMC by Andrieu and Roberts (2009). Here I provide an alternative perspective, where I describe MCMC-ABC as pseudo-marginal MCMC.

Compared to rejection ABC, which proposes from the prior and thus can lead to most proposals being rejected, MCMC-ABC tends to accept more often. As discussed earlier, we can think of

Algorithm 3.4: Markov-chain Monte Carlo ABC

Propose $\boldsymbol{\theta}' \sim q(\boldsymbol{\theta}' \,|\, \boldsymbol{\theta})$
Simulate $\mathbf{x} \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}')$
if $\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon$ **then**
    Calculate acceptance ratio $\alpha = \frac{p(\boldsymbol{\theta}')\, q(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}')}{p(\boldsymbol{\theta})\, q(\boldsymbol{\theta}' \,|\, \boldsymbol{\theta})}$
    Sample $u \sim \mathcal{U}(0, 1)$
    if $u \leq \alpha$ **then**
        return $\boldsymbol{\theta}'$
    else
        return $\boldsymbol{\theta}$
else
    return $\boldsymbol{\theta}$

the proposal $q(\boldsymbol{\theta}' \,|\, \boldsymbol{\theta})$ as randomly 'perturbing' accepted parameters $\boldsymbol{\theta}$ in order to propose new parameters $\boldsymbol{\theta}'$; if the perturbation is small, the proposed parameters are also likely to be accepted. However, unlike rejection ABC which produces independent samples, MCMC-ABC produces dependent samples, which can lead to low effective sample size.

Similarly to rejection ABC, the acceptance probability of MCMC-ABC vanishes as ϵ becomes small or as the data dimensionality becomes large. In practice, the data is typically transformed into summary statistics in order to reduce the data dimensionality. Another issue with MCMC-ABC is the initialization of the Markov chain: if the initial parameters $\boldsymbol{\theta}$ are unlikely to generate data near $\mathbf{x}_o$, the chain may be stuck at its initial state for a long time. In practice, one possibility is to run rejection ABC until one parameter sample is accepted, and initialize the Markov chain with that parameter sample.

3.2.3 Sequential Monte Carlo ABC

As we have seen, the low acceptance rate of rejection ABC is partly due to parameters being proposed from the prior $p(\boldsymbol{\theta})$. Indeed, if the approximate posterior $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$ is significantly narrower than the prior, it's highly unlikely that a prior sample will be accepted. In the previous section, I discussed an approach for increasing the acceptance rate based on perturbing accepted parameters using Markov-chain Monte Carlo. In this section, I will discuss an alternative approach based on importance sampling.

To begin with, consider a variant of rejection ABC where instead of proposing parameters from the prior $p(\boldsymbol{\theta})$, we propose parameters from an alternative distribution $\tilde{p}(\boldsymbol{\theta})$. Parameter samples accepted under this procedure will be distributed according to the following distribution:

$$\tilde{p}(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon) \propto \Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta})\, \tilde{p}(\boldsymbol{\theta}). \tag{3.34}$$

Assuming $\tilde{p}(\boldsymbol{\theta})$ is non-zero in the support of the prior, we can approximate $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$ with a population of N importance-weighted samples $\{(w_1, \boldsymbol{\theta}_1), \ldots, (w_N, \boldsymbol{\theta}_N)\}$, where each $\boldsymbol{\theta}_n$ is obtained using the above procedure, and $\{w_1, \ldots, w_N\}$ are importance weights that are normalized to sum to 1 and are calculated by:

$$w_n \propto \frac{p(\boldsymbol{\theta}_n \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)}{\tilde{p}(\boldsymbol{\theta}_n \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)} \propto \frac{\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta}_n)\, p(\boldsymbol{\theta}_n)}{\Pr(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon \,|\, \boldsymbol{\theta}_n)\, \tilde{p}(\boldsymbol{\theta}_n)} = \frac{p(\boldsymbol{\theta}_n)}{\tilde{p}(\boldsymbol{\theta}_n)}. \tag{3.35}$$

The above procedure, which I refer to as *importance-sampling ABC*, is shown in algorithm 3.5. Importance-sampling ABC reduces to rejection ABC if $\tilde{p}(\boldsymbol{\theta}) = p(\boldsymbol{\theta})$.

Algorithm 3.5: Importance-sampling ABC

for $n = 1 : N$ **do**
  repeat
    Sample $\boldsymbol{\theta}_n \sim \tilde{p}(\boldsymbol{\theta})$
    Simulate $\mathbf{x} \sim p(\mathbf{x} \mid \boldsymbol{\theta}_n)$
  until $\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon$
Calculate importance weights $w_n \propto \frac{p(\boldsymbol{\theta}_n)}{\tilde{p}(\boldsymbol{\theta}_n)}$ such that they sum to 1
return $\{(w_1, \boldsymbol{\theta}_1), \ldots, (w_N, \boldsymbol{\theta}_N)\}$

The efficiency of importance-sampling ABC depends heavily on the choice of the proposal $\tilde{p}(\boldsymbol{\theta})$. If the proposal is too broad the acceptance rate will be low, whereas if the proposal is too narrow the importance weights will have high variance (see Alsing et al., 2018b, for a discussion on the optimality of the proposal). In practice, one approach for constructing a proposal is to perform a preliminary run of importance-sampling ABC with $\epsilon' > \epsilon$, obtain a population of importance-weighted parameter samples $\{(w_1, \boldsymbol{\theta}_1), \ldots, (w_N, \boldsymbol{\theta}_N)\}$, and take $\tilde{p}(\boldsymbol{\theta})$ to be a mixture distribution defined by:

$$\tilde{p}(\boldsymbol{\theta}) = \sum_n w_n \, q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_n). \tag{3.36}$$

Similar to MCMC-ABC, $q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_n)$ can be thought of as a way to perturb $\boldsymbol{\theta}_n$. For instance, a common choice is to take $q(\boldsymbol{\theta} \mid \boldsymbol{\theta}_n)$ to be a Gaussian distribution centred at $\boldsymbol{\theta}_n$, which corresponds to adding zero-mean Gaussian noise to $\boldsymbol{\theta}_n$.

Based on the above idea, we can run a sequence of T rounds of importance-sampling ABC with $\epsilon_1 > \ldots > \epsilon_T$, and construct the proposal for round t using the importance-weighted population obtained in round $t-1$. The first round can use the prior $p(\boldsymbol{\theta})$ as proposal and a large enough ϵ_1 so that the acceptance rate is not too low. This procedure is known as *sequential Monte Carlo ABC*, and is shown in algorithm 3.6. Further discussion on SMC-ABC and different variants of the algorithm are provided by Beaumont et al. (2009); Bonassi and West (2015); Sisson et al. (2007); Toni et al. (2009).

An issue with sequential Monte Carlo more broadly (and not just in the context of ABC) is *sample degeneracy*, which occurs when only few samples have non-negligible weights. If that's the case, the effective sample size, i.e. the number of independent samples that would be equivalent to the weighted population, is significantly less than the population size N. One strategy for avoiding sample degeneracy, which is shown in algorithm 3.6, is to estimate the effective sample size after each round of importance-sampling ABC; if the estimated effective sample size is less than a threshold (e.g. $N/2$), we resample the population with probabilities given by the weights in order to obtain a new population with equal weights. In the context of ABC in particular, the above resampling may not be necessary, since sampling the next population from the proposal distribution already diversifies its samples. In any case, it's a good idea to monitor the effective sample size of each population as the algorithm progresses, as a consistently low sample size can serve as a warning of potentially poor performance. In practice, the effective sample size S of a weighted population $\{(w_1, \boldsymbol{\theta}_1), \ldots, (w_N, \boldsymbol{\theta}_N)\}$ can be estimated by:

$$\hat{S} = \frac{1}{\sum_n w_n^2}. \tag{3.37}$$

A justification of the above estimate is provided by Nowozin (2015).

Algorithm 3.6: Sequential Monte Carlo ABC

Run rejection ABC with tolerance ϵ_1
Obtain initial population $\left\{\boldsymbol{\theta}_1^{(1)}, \ldots, \boldsymbol{\theta}_N^{(1)}\right\}$
Set initial weights $w_n^{(1)} \propto 1$
for $t = 2 : T$ **do**
  Set proposal $\tilde{p}(\boldsymbol{\theta}) = \sum_n w_n^{(t-1)}\, q\left(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}_n^{(t-1)}\right)$
  Run importance-sampling ABC with tolerance ϵ_t and proposal $\tilde{p}(\boldsymbol{\theta})$
  Obtain weighted population $\left\{\left(w_1^{(t)}, \boldsymbol{\theta}_1^{(t)}\right), \ldots, \left(w_N^{(t)}, \boldsymbol{\theta}_N^{(t)}\right)\right\}$
  Estimate effective sample size $\hat{S} = \left(\sum_n w_n^2\right)^{-1}$
  if $\hat{S} < S_{\min}$ **then**
    Resample $\left\{\boldsymbol{\theta}_1^{(t)}, \ldots, \boldsymbol{\theta}_N^{(t)}\right\}$ with probabilities $\left\{w_1^{(t)}, \ldots, w_N^{(t)}\right\}$
    Set weights $w_n^{(t)} \propto 1$
return $\left\{\left(w_1^{(T)}, \boldsymbol{\theta}_1^{(T)}\right), \ldots, \left(w_N^{(T)}, \boldsymbol{\theta}_N^{(T)}\right)\right\}$

SMC-ABC is a strong baseline for likelihood-free inference; its acceptance rate is typically higher than rejection ABC, and it can be easier to tune than MCMC-ABC. One way to use SMC-ABC in practice is to start with a tolerance ϵ_1 that is large enough for rejection ABC to be practical, and exponentially decay it after each round in order to obtain increasingly accurate posterior samples. Nevertheless, even though SMC-ABC can produce a good proposal distribution after a few rounds, it doesn't solve the problem of the acceptance probability eventually vanishing as ϵ becomes small. In practice, as ϵ decreases in every round, the required number of simulations increases, and can thus become prohibitively large after a few rounds.

3.3 The paper

This section presents the paper *Fast ϵ-free Inference of Simulator Models with Bayesian Conditional Density Estimation*, which is the main contribution of this chapter. In the paper, we cast likelihood-free inference as density-estimation problem: we estimate the posterior density using a neural density estimator trained on data simulated from the model. We employ a Bayesian mixture-density network as our neural density estimator, and we describe how to use the network in order to guide simulations and thus accelerate the training process.

The paper was first published as a preprint on arXiv in May 2016. Afterwards, it was accepted for publication at the conference *Advances in Neural Information Processing Systems (NeurIPS)* in December 2016. There were about 2,500 submissions to NeurIPS in 2016, of which 568 were accepted for publication.

Author contributions

The paper is co-authored by me and Iain Murray. We jointly conceived and developed the method. As the leading author, I wrote the code, performed the experiments, and drafted the paper. Iain Murray supervised the project, and revised the final draft.

Differences in notation

For precision, in this chapter I use $p(\cdot)$ for probability densities and $\Pr(\cdot)$ for probability measures. The paper doesn't make this distinction; instead, it uses the term 'probability' for both probability densities and probability measures, and denotes both of them by $p(\cdot)$ or similar. This overloaded notation, common in the machine-learning literature, should be interpreted appropriately depending on context.

Throughout this chapter I use $\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon$ to denote the acceptance condition of ABC; the paper uses $\|\mathbf{x} - \mathbf{x}_o\| < \epsilon$ instead. The difference is immaterial, since the set $\{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_o\| = \epsilon\}$ has zero probability measure by assumption.

Finally, the paper uses $\langle f(\mathbf{x}) \rangle_{p(\mathbf{x})}$ to denote the expectation of a function $f(\mathbf{x})$ with respect to a distribution $p(\mathbf{x})$, whereas the rest of the thesis uses $\mathbb{E}_{p(\mathbf{x})}(f(\mathbf{x}))$.

Minor corrections

Section 2.1 of the paper states that $\mathbf{x} = \mathbf{x}_o$ corresponds to setting $\epsilon = 0$ in $\|\mathbf{x} - \mathbf{x}_o\| < \epsilon$. Strictly speaking, this statement is incorrect, since $\{\mathbf{x} : \|\mathbf{x} - \mathbf{x}_o\| < 0\}$ is the empty set. For this reason, in this chapter I use $\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon$, such that setting ϵ to zero correctly implies $\mathbf{x} = \mathbf{x}_o$.

Section 4 of the paper misrepresents the work of Fan et al. (2013) as a synthetic likelihood method that trains "a separate density model of $\mathbf{x}$ for each $\boldsymbol{\theta}$ by repeatedly simulating the model for fixed $\boldsymbol{\theta}$". This is incorrect; in reality, Fan et al. (2013) use a single conditional-density model of $\mathbf{x}$ given $\boldsymbol{\theta}$ that is trained on pairs $(\boldsymbol{\theta}, \mathbf{x})$ independently simulated from a joint density $r(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})$, where $r(\boldsymbol{\theta})$ is some reference density.

Fast ϵ-free Inference of Simulation Models with Bayesian Conditional Density Estimation

George Papamakarios
School of Informatics
University of Edinburgh
g.papamakarios@ed.ac.uk

Iain Murray
School of Informatics
University of Edinburgh
i.murray@ed.ac.uk

Abstract

Many statistical models can be simulated forwards but have intractable likelihoods. Approximate Bayesian Computation (ABC) methods are used to infer properties of these models from data. Traditionally these methods approximate the posterior over parameters by conditioning on data being inside an ϵ-ball around the observed data, which is only correct in the limit $\epsilon \to 0$. Monte Carlo methods can then draw samples from the approximate posterior to approximate predictions or error bars on parameters. These algorithms critically slow down as $\epsilon \to 0$, and in practice draw samples from a broader distribution than the posterior. We propose a new approach to likelihood-free inference based on Bayesian conditional density estimation. Preliminary inferences based on limited simulation data are used to guide later simulations. In some cases, learning an accurate parametric representation of the entire true posterior distribution requires fewer model simulations than Monte Carlo ABC methods need to produce a single sample from an approximate posterior.

1 Introduction

A simulator-based model is a data-generating process described by a computer program, usually with some free parameters we need to learn from data. Simulator-based modelling lends itself naturally to scientific domains such as evolutionary biology [1], ecology [33], disease epidemics [11], economics [9] and cosmology [26], where observations are best understood as products of underlying physical processes. Inference in these models amounts to discovering plausible parameter settings that could have generated our observed data. The application domains mentioned can require properly calibrated distributions that express uncertainty over plausible parameters, rather than just point estimates, in order to reach scientific conclusions or make decisions.

As an analytical expression for the likelihood of parameters given observations is typically not available for simulator-based models, conventional likelihood-based Bayesian inference is not applicable. An alternative family of algorithms for likelihood-free inference has been developed, referred to as Approximate Bayesian Computation (ABC). These algorithms simulate the model repeatedly and only accept parameter settings which generate synthetic data similar to the observed data, typically gathered in a real-world experiment.

Rejection ABC [24], the most basic ABC algorithm, simulates the model for each setting of proposed parameters, and rejects parameters if the generated data is not within a certain distance from the observations. The accepted parameters form a set of independent samples from an approximate posterior. Markov Chain Monte Carlo ABC (MCMC-ABC) [15] is an improvement over rejection ABC which, instead of independently proposing parameters, explores the parameter space by perturbing the most recently accepted parameters. Sequential Monte Carlo ABC (SMC-ABC) [2, 5] uses importance sampling to simulate a sequence of slowly-changing distributions, the last of which is an approximation to the parameter posterior.

Conventional ABC algorithms such as the above suffer from three drawbacks. First, they only represent the parameter posterior as a set of (possibly weighted or correlated) samples. A sample-based representation easily gives estimates and error bars of individual parameters, and model predictions. However these computations are noisy, and it is not obvious how to perform some other computations using samples, such as combining posteriors from two separate analyses. Second, the parameter samples do not come from the correct Bayesian posterior, but from an approximation based on assuming a pseudo-observation that the data is within an ϵ-ball centred on the data actually observed. Third, as the ϵ-tolerance is reduced, it can become impractical to simulate the model enough times to match the observed data even once. When simulations are expensive to perform, good quality inference becomes impractical.

We propose a parametric approach to likelihood-free inference, which unlike conventional ABC does not suffer from the above three issues. Instead of returning samples from an ϵ-approximation to the posterior, our approach learns a parametric approximation to the exact posterior, which can be made as accurate as required. Preliminary fits to the posterior are used to guide future simulations, which can reduce the number of simulations required to learn an accurate approximation by orders of magnitude. Our approach uses conditional density estimation with Bayesian neural networks, and draws upon advances in parametric density estimation, stochastic variational inference, and recognition networks, as discussed in the related work section.

2 Bayesian conditional density estimation for likelihood-free inference

2.1 Simulator-based models and ABC

Let $\boldsymbol{\theta}$ be a vector of parameters controlling a simulator-based model, and let $\mathbf{x}$ be a data vector generated by the model. The model may be provided as a probabilistic program that can be easily simulated, and implicitly defines a likelihood $p(\mathbf{x} \,|\, \boldsymbol{\theta})$, which we assume we cannot evaluate. Let $p(\boldsymbol{\theta})$ encode our prior beliefs about the parameters. Given an observation $\mathbf{x}_o$, we are interested in the parameter posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) \propto p(\mathbf{x} = \mathbf{x}_o \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})$.

As the likelihood $p(\mathbf{x} = \mathbf{x}_o \,|\, \boldsymbol{\theta})$ is unavailable, conventional Bayesian inference cannot be carried out. The principle behind ABC is to approximate $p(\mathbf{x} = \mathbf{x}_o \,|\, \boldsymbol{\theta})$ by $p(\|\mathbf{x} - \mathbf{x}_o\| < \epsilon \,|\, \boldsymbol{\theta})$ for a sufficiently small value of ϵ, and then estimate the latter—e.g. by Monte Carlo—using simulations from the model. Hence, ABC approximates the posterior by $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| < \epsilon)$, which is typically broader and more uncertain. ABC can trade off computation for accuracy by decreasing ϵ, which improves the approximation to the posterior but requires more simulations from the model. However, the approximation becomes exact only when $\epsilon \to 0$, in which case simulations never match the observations, $p(\|\mathbf{x} - \mathbf{x}_o\| < \epsilon \,|\, \boldsymbol{\theta}) \to 0$, and existing methods break down. In this paper, we refer to $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ as the *exact posterior*, as it corresponds to setting $\epsilon = 0$ in $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| < \epsilon)$.

In most practical applications of ABC, $\mathbf{x}$ is taken to be a fixed-length vector of summary statistics that is calculated from data generated by the simulator, rather than the raw data itself. Extracting statistics is often necessary in practice, to reduce the dimensionality of the data and maintain $p(\|\mathbf{x} - \mathbf{x}_o\| < \epsilon \,|\, \boldsymbol{\theta})$ to practically acceptable levels. For the purposes of this paper, we will make no distinction between raw data and summary statistics, and we will regard the calculation of summary statistics as part of the data generating process.

2.2 Learning the posterior

Rather than using simulations from the model in order to estimate an approximate likelihood, $p(\|\mathbf{x} - \mathbf{x}_o\| < \epsilon \,|\, \boldsymbol{\theta})$, we will use the simulations to directly estimate $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$. We will run simulations for parameters drawn from a distribution, $\tilde{p}(\boldsymbol{\theta})$, which we shall refer to as the *proposal prior*. The proposition below indicates how we can then form a consistent estimate of the exact posterior, using a flexible family of conditional densities, $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$, parameterized by a vector $\boldsymbol{\phi}$.

Proposition 1. *We assume that each of a set of N pairs $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ was independently generated by*

$$\boldsymbol{\theta}_n \sim \tilde{p}(\boldsymbol{\theta}) \quad \textit{and} \quad \mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}_n). \tag{1}$$

In the limit $N \to \infty$, the probability of the parameter vectors $\prod_n q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n)$ is maximized w.r.t. $\boldsymbol{\phi}$ if and only if

$$q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) \propto \frac{\tilde{p}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}\, p(\boldsymbol{\theta} \,|\, \mathbf{x}), \tag{2}$$

provided a setting of $\boldsymbol{\phi}$ that makes $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x})$ proportional to $\frac{\tilde{p}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}\, p(\boldsymbol{\theta} \mid \mathbf{x})$ exists.

Intuition: if we simulated enough parameters from the prior, the density estimator $q_{\boldsymbol{\phi}}$ would learn a conditional of the joint prior model over parameters and data, which is the posterior $p(\boldsymbol{\theta} \mid \mathbf{x})$. If we simulate parameters drawn from another distribution, we need to "importance reweight" the result. A more detailed proof can be found in Appendix A.

The proposition above suggests the following procedure for learning the posterior: (a) propose a set of parameter vectors $\{\boldsymbol{\theta}_n\}$ from the proposal prior; (b) for each $\boldsymbol{\theta}_n$ run the simulator to obtain a corresponding data vector $\mathbf{x}_n$; (c) train $q_{\boldsymbol{\phi}}$ with maximum likelihood on $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}$; and (d) estimate the posterior by

$$\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o) \propto \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})}\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x}_o). \tag{3}$$

This procedure is summarized in Algorithm 2.

2.3 Choice of conditional density estimator and proposal prior

In choosing the types of density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x})$ and proposal prior $\tilde{p}(\boldsymbol{\theta})$, we need to meet the following criteria: (a) $q_{\boldsymbol{\phi}}$ should be flexible enough to represent the posterior but easy to train with maximum likelihood; (b) $\tilde{p}(\boldsymbol{\theta})$ should be easy to evaluate and sample from; and (c) the right-hand side expression in Equation (3) should be easily evaluated and normalized.

We draw upon work on conditional neural density estimation and take $q_{\boldsymbol{\phi}}$ to be a Mixture Density Network (MDN) [3] with fully parameterized covariance matrices. That is, $q_{\boldsymbol{\phi}}$ takes the form of a mixture of K Gaussian components $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x}) = \sum_k \alpha_k\, \mathcal{N}(\boldsymbol{\theta} \mid \mathbf{m}_k, \mathbf{S}_k)$, whose mixing coefficients $\{\alpha_k\}$, means $\{\mathbf{m}_k\}$ and covariance matrices $\{\mathbf{S}_k\}$ are computed by a feedforward neural network parameterized by $\boldsymbol{\phi}$, taking $\mathbf{x}$ as input. Such an architecture is capable of representing any conditional distribution arbitrarily accurately—provided the number of components K and number of hidden units in the neural network are sufficiently large—while remaining trainable by backpropagation. The parameterization of the MDN is detailed in Appendix B.

We take the proposal prior to be a single Gaussian $\tilde{p}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \mid \mathbf{m}_0, \mathbf{S}_0)$, with mean $\mathbf{m}_0$ and full covariance matrix $\mathbf{S}_0$. Assuming the prior $p(\boldsymbol{\theta})$ is a simple distribution (uniform or Gaussian, as is typically the case in practice), then this choice allows us to calculate $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$ in Equation (3) analytically. That is, $\hat{p}(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$ will be a mixture of K Gaussians, whose parameters will be a function of $\{\alpha_k, \mathbf{m}_k, \mathbf{S}_k\}$ evaluated at $\mathbf{x}_o$ (as detailed in Appendix C).

2.4 Learning the proposal prior

Simple rejection ABC is inefficient because the posterior $p(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$ is typically much narrower than the prior $p(\boldsymbol{\theta})$. A parameter vector $\boldsymbol{\theta}$ sampled from $p(\boldsymbol{\theta})$ will rarely be plausible under $p(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$ and will most likely be rejected. Practical ABC algorithms attempt to reduce the number of rejections by modifying the way they propose parameters; for instance, MCMC-ABC and SMC-ABC propose new parameters by perturbing parameters they already consider plausible, in the hope that nearby parameters remain plausible.

In our framework, the key to efficient use of simulations lies in the choice of proposal prior. If we take $\tilde{p}(\boldsymbol{\theta})$ to be the actual prior, then $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x})$ will learn the posterior for all $\mathbf{x}$, as can be seen from Equation (2). Such a strategy however is grossly inefficient if we are only interested in the posterior for $\mathbf{x} = \mathbf{x}_o$. Conversely, if $\tilde{p}(\boldsymbol{\theta})$ closely matches $p(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$, then most simulations will produce samples that are highly informative in learning $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x})$ for $\mathbf{x} = \mathbf{x}_o$. In other words, if we already knew the true posterior, we could use it to construct an efficient proposal prior for learning it.

We exploit this idea to set up a fixed-point system. Our strategy becomes to learn an efficient proposal prior that closely approximates the posterior as follows: (a) initially take $\tilde{p}(\boldsymbol{\theta})$ to be the prior $p(\boldsymbol{\theta})$; (b) propose N samples $\{\boldsymbol{\theta}_n\}$ from $\tilde{p}(\boldsymbol{\theta})$ and corresponding samples $\{\mathbf{x}_n\}$ from the simulator, and train $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x})$ on them; (c) approximate the posterior using Equation (3) and set $\tilde{p}(\boldsymbol{\theta})$ to it; (d) repeat until $\tilde{p}(\boldsymbol{\theta})$ has converged. This procedure is summarized in Algorithm 1.

In the procedure above, as long as $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \mid \mathbf{x})$ has only one Gaussian component ($K = 1$) then $\tilde{p}(\boldsymbol{\theta})$ remains a single Gaussian throughout. Moreover, in each iteration we initialize $q_{\boldsymbol{\phi}}$ with the density

Algorithm 1: Training of proposal prior

initialize $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ with one component
$\tilde{p}(\boldsymbol{\theta}) \leftarrow p(\boldsymbol{\theta})$
repeat
  for $n = 1..N$ **do**
    sample $\boldsymbol{\theta}_n \sim \tilde{p}(\boldsymbol{\theta})$
    sample $\mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}_n)$
  end
  retrain $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ on $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}$
  $\tilde{p}(\boldsymbol{\theta}) \leftarrow \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})}\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$
until $\tilde{p}(\boldsymbol{\theta})$ *has converged*;

Algorithm 2: Training of posterior

initialize $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ with K components
`// if` $q_{\boldsymbol{\phi}}$ `available by Algorithm 1`
`// initialize by replicating its`
`// one component` K `times`
for $n = 1..N$ **do**
  sample $\boldsymbol{\theta}_n \sim \tilde{p}(\boldsymbol{\theta})$
  sample $\mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}_n)$
end
train $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ on $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}$
$\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) \leftarrow \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})}\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$

estimator learnt in the iteration before, thus we keep training $q_{\boldsymbol{\phi}}$ throughout. This initialization allows us to use a small sample size N in each iteration, thus making efficient use of simulations.

As we shall demonstrate in Section 3, the procedure above learns Gaussian approximations to the true posterior fast: in our experiments typically 4–6 iterations of 200–500 samples each were sufficient. This Gaussian approximation can be used as a rough but cheap approximation to the true posterior, or it can serve as a good proposal prior in Algorithm 2 for efficiently fine-tuning a non-Gaussian multi-component posterior. If the second strategy is adopted, then we can reuse the single-component neural density estimator learnt in Algorithm 1 to initialize $q_{\boldsymbol{\phi}}$ in Algorithm 2. The weights in the final layer of the MDN are replicated K times, with small random perturbations to break symmetry.

2.5 Use of Bayesian neural density estimators

To make Algorithm 1 as efficient as possible, the number of simulations per iteration N should be kept small, while at the same time it should provide a sufficient training signal for $q_{\boldsymbol{\phi}}$. With a conventional MDN, if N is made too small, there is a danger of overfitting, especially in early iterations, leading to over-confident proposal priors and an unstable procedure. Early stopping could be used to avoid overfitting; however a significant fraction of the N samples would have to be used as a validation set, leading to inefficient use of simulations.

As a better alternative, we developed a Bayesian version of the MDN using Stochastic Variational Inference (SVI) for neural networks [13]. We shall refer to this Bayesian version of the MDN as MDN-SVI. An MDN-SVI has two sets of adjustable parameters of the same size, the means $\boldsymbol{\phi}_m$ and the log variances $\boldsymbol{\phi}_s$. The means correspond to the parameters $\boldsymbol{\phi}$ of a conventional MDN. During training, Gaussian noise of variance $\exp \boldsymbol{\phi}_s$ is added to the means independently for each training example $(\boldsymbol{\theta}_n, \mathbf{x}_n)$. The Bayesian interpretation of this procedure is that it optimizes a variational Gaussian posterior with a diagonal covariance matrix over parameters $\boldsymbol{\phi}$. At prediction time, the noise is switched off and the MDN-SVI behaves like a conventional MDN with $\boldsymbol{\phi} = \boldsymbol{\phi}_m$. Appendix D details the implementation and training of MDN-SVI. We found that using an MDN-SVI instead of an MDN improves the robustness and efficiency of Algorithm 1 because (a) MDN-SVI is resistant to overfitting, allowing us to use a smaller number of simulations N; (b) no validation set is needed, so all samples can be used for training; and (c) since overfitting is not an issue, no careful tuning of training time is necessary.

3 Experiments

We showcase three versions of our approach: (a) learning the posterior with Algorithm 2 where $q_{\boldsymbol{\phi}}$ is a conventional MDN and the proposal prior $\tilde{p}(\boldsymbol{\theta})$ is taken to be the actual prior $p(\boldsymbol{\theta})$, which we refer to as *MDN with prior*; (b) training a proposal prior with Algorithm 1 where $q_{\boldsymbol{\phi}}$ is an MDN-SVI, which we refer to as *proposal prior*; and (c) learning the posterior with Algorithm 2 where $q_{\boldsymbol{\phi}}$ is an MDN-SVI and the proposal prior $\tilde{p}(\boldsymbol{\theta})$ is taken to be the one learnt in (b), which we refer to as *MDN with proposal*. All MDNs were trained using Adam [12] with its default parameters.

We compare to three ABC baselines: (a) rejection ABC [24], where parameters are proposed from the prior and are accepted if $\|\mathbf{x} - \mathbf{x}_o\| < \epsilon$; (b) MCMC-ABC [15] with a spherical Gaussian proposal, whose variance we manually tuned separately in each case for best performance; and (c) SMC-ABC

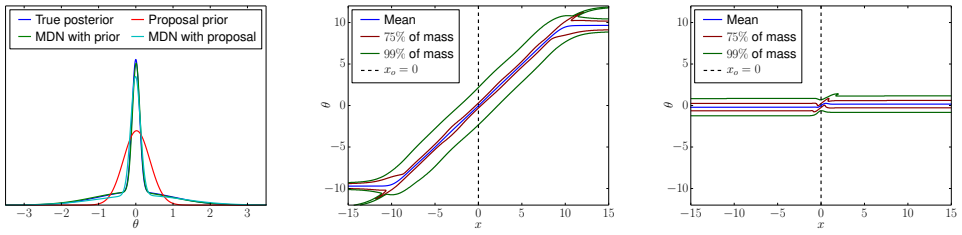


Figure 1: Results on mixture of two Gaussians. **Left**: approximate posteriors learnt by each strategy for $x_o = 0$. **Middle**: full conditional density $q_{\boldsymbol{\phi}}(\theta|x)$ leant by the MDN trained with prior. **Right**: full conditional density $q_{\boldsymbol{\phi}}(\theta|x)$ learnt by the MDN-SVI trained with proposal prior. Vertical dashed lines show the location of the observation $x_o = 0$.

[2], where the sequence of ϵ's was exponentially decayed, with a decay rate manually tuned separately in each case for best performance. MCMC-ABC was given the unrealistic advantage of being initialized with a sample from rejection ABC, removing the need for an otherwise necessary burn-in period. Code for reproducing the experiments is provided at `https://github.com/gpapamak/epsilon_free_inference`.

3.1 Mixture of two Gaussians

The first experiment is a toy problem where the goal is to infer the common mean θ of a mixture of two 1D Gaussians, given a single datapoint x_o. The setup is

$$p(\theta) = \mathcal{U}(\theta \,|\, \theta_\alpha, \theta_\beta) \quad \text{and} \quad p(x \,|\, \theta) = \alpha\, \mathcal{N}\big(x \,|\, \theta, \sigma_1^2\big) + (1-\alpha)\, \mathcal{N}\big(x \,|\, \theta, \sigma_2^2\big), \tag{4}$$

where $\theta_\alpha = -10$, $\theta_\beta = 10$, $\alpha = 0.5$, $\sigma_1 = 1$, $\sigma_2 = 0.1$ and $x_o = 0$. The posterior can be calculated analytically, and is proportional to an equal mixture of two Gaussians centred at x_o with variances σ_1^2 and σ_2^2, restricted to $[\theta_\alpha, \theta_\beta]$. This problem is often used in the SMC-ABC literature to illustrate the difficulty of MCMC-ABC in representing long tails. Here we use it to demonstrate the correctness of our approach and its ability to accurately represent non-Gaussian long-tailed posteriors.

Figure 1 shows the results of neural density estimation using each strategy. All MDNs have one hidden layer with 20 tanh units and 2 Gaussian components, except for the proposal prior MDN which has a single component. Both MDN with prior and MDN with proposal learn good parametric approximations to the true posterior, and the proposal prior is a good Gaussian approximation to it. We used 10K simulations to train the MDN with prior, whereas the prior proposal took 4 iterations of 200 simulations each to train, and the MDN with proposal took 1000 simulations on top of the previous 800. The MDN with prior learns the posterior distributions for a large range of possible observations x (middle plot of Figure 1), whereas the MDN with proposal gives accurate posterior probabilities only near the value actually observed (right plot of Figure 1).

3.2 Bayesian linear regression

In Bayesian linear regression, the goal is to infer the parameters $\boldsymbol{\theta}$ of a linear map from noisy observations of outputs at known inputs. The setup is

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \,|\, \mathbf{m}, \mathbf{S}) \quad \text{and} \quad p(\mathbf{x} \,|\, \boldsymbol{\theta}) = \textstyle\prod_i \mathcal{N}\big(x_i \,|\, \boldsymbol{\theta}^T \mathbf{u}_i, \sigma^2\big), \tag{5}$$

where we took $\mathbf{m} = \mathbf{0}$, $\mathbf{S} = \mathbf{I}$, $\sigma = 0.1$, randomly generated inputs $\{\mathbf{u}_i\}$ from a standard Gaussian, and randomly generated observations $\mathbf{x}_o$ from the model. In our setup, $\boldsymbol{\theta}$ and $\mathbf{x}$ have 6 and 10 dimensions respectively. The posterior is analytically tractable, and is a single Gaussian.

All MDNs have one hidden layer of 50 tanh units and one Gaussian component. ABC methods were run for a sequence of decreasing ϵ's, up to their failing points. To measure the approximation quality to the posterior, we analytically calculated the KL divergence from the true posterior to the learnt posterior (which for ABC was taken to be a Gaussian fit to the set of returned posterior samples). The left of Figure 2 shows the approximation quality vs ϵ; MDN methods are shown as horizontal

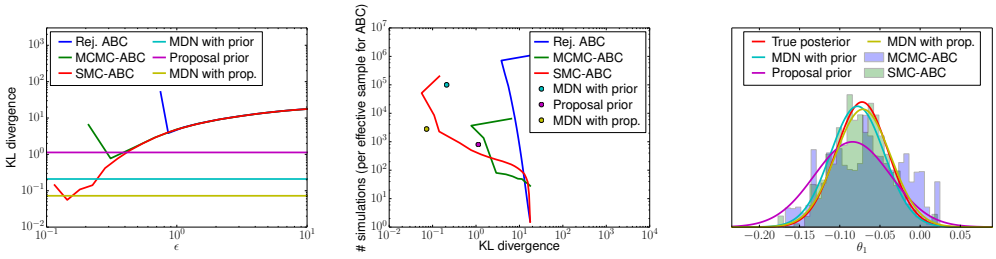


Figure 2: Results on Bayesian linear regression. **Left**: KL divergence from true posterior to approximation vs ϵ; lower is better. **Middle**: number of simulations vs KL divergence; lower left is better. Note that number of simulations is total for MDNs, and per effective sample for ABC. **Right**: Posterior marginals for θ_1 as computed by each method. ABC posteriors (represented as histograms) correspond to the setting of ϵ that minimizes the KL in the left plot.

lines. As ϵ is decreased, ABC methods sample from an increasingly better approximation to the true posterior, however they eventually reach their failing point, or take prohibitively long. The best approximations are achieved by MDN with proposal and a very long run of SMC-ABC.

The middle of Figure 2 shows the increase in number of simulations needed to improve approximation quality (as ϵ decreases). We quote the *total* number of simulations for MDN training, and the number of simulations *per effective sample* for ABC. Appendix E describes how the number of effective samples is calculated. The number of simulations per effective sample should be multiplied by the number of effective samples needed in practice. Moreover, SMC-ABC will not work well with only one particle, so many times the quoted cost will always be needed. Here, MDNs make more efficient use of simulations than Monte Carlo ABC methods. Sequentially fitting a prior proposal was more than ten times cheaper than training with prior samples, and more accurate.

3.3 Lotka–Volterra predator-prey population model

The Lotka–Volterra model is a stochastic Markov jump process that describes the continuous time evolution of a population of predators interacting with a population of prey. There are four possible reactions: (a) a predator being born, (b) a predator dying, (c) a prey being born, and (d) a prey being eaten by a predator. Positive parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)$ control the rate of each reaction. Given a set of statistics $\mathbf{x}_o$ calculated from an observed population time series, the objective is to infer $\boldsymbol{\theta}$. We used a flat prior over $\log \boldsymbol{\theta}$, and calculated a set of 9 statistics $\mathbf{x}$. The full setup is detailed in Appendix F. The Lotka–Volterra model is commonly used in the ABC literature as a realistic model which can be simulated, but whose likelihood is intractable. One of the properties of Lotka–Volterra is that typical nature-like observations only occur for very specific parameter settings, resulting in narrow, Gaussian-like posteriors that are hard to recover.

The MDN trained with prior has two hidden layers of 50 tanh units each, whereas the MDN-SVI used to train the proposal prior and the MDN-SVI trained with proposal have one hidden layer of 50 tanh units. All three have one Gaussian component. We found that using more than one components made no difference to the results; in all cases the MDNs chose to use only one component and switch the rest off, which is consistent with our observation about the near-Gaussianity of the posterior.

We measure how well each method retrieves the true parameter values that were used to generate $\mathbf{x}_o$ by calculating their log probability under each learnt posterior; for ABC a Gaussian fit to the posterior samples was used. The left panel of Figure 3 shows how this log probability varies with ϵ, demonstrating the superiority of MDN methods over ABC. In the middle panel we can see that MDN training with proposal makes efficient use of simulations compared to training with prior and ABC; note that for ABC the number of simulations is only for *one effective sample*. In the right panel, we can see that the estimates returned by MDN methods are more confident around the true parameters compared to ABC, because the MDNs learn the exact posterior rather than an inflated version of it like ABC does (plots for the other three parameters look similar).

We found that when training an MDN with a well-tuned proposal that focuses on the plausible region, an MDN with fewer parameters is needed compared to training with the prior. This is because the

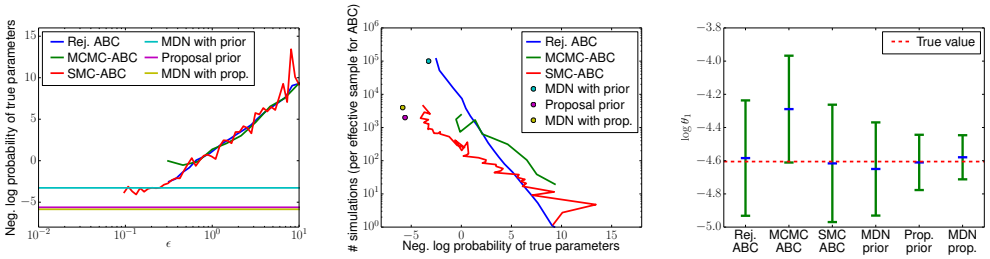


Figure 3: Results on Lotka–Volterra. **Left**: negative log probability of true parameters vs ϵ; lower is better. **Middle**: number of simulations vs negative log probability; lower left is better. Note that number of simulations is total for MDNs, but per effective sample for ABC. **Right**: Estimates of $\log \theta_1$ with 2 standard deviations. ABC estimates used many more simulations with the smallest feasible ϵ.

MDN trained with proposal needs to learn only the *local* relationship between $\mathbf{x}$ and $\boldsymbol{\theta}$ near $\mathbf{x}_o$, as opposed to in the entire domain of the prior. Hence, not only are savings achieved in number of simulations, but also training the MDN itself becomes more efficient.

3.4 M/G/1 queue model

The M/G/1 queue model describes the processing of a queue of continuously arriving jobs by a single server. In this model, the time the server takes to process each job is independently and uniformly distributed in the interval $[\theta_1, \theta_2]$. The time interval between arrival of two consecutive jobs is independently and exponentially distributed with rate θ_3. The server observes only the time intervals between departure of two consecutive jobs. Given a set of equally-spaced percentiles $\mathbf{x}_o$ of inter-departure times, the task is to infer parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$. This model is easy to simulate but its likelihood is intractable, and it has often been used as an ABC benchmark [4, 18]. Unlike Lotka–Volterra, data $\mathbf{x}$ is weakly informative about $\boldsymbol{\theta}$, and hence the posterior over $\boldsymbol{\theta}$ tends to be broad and non-Gaussian. In our setup, we placed flat independent priors over θ_1, $\theta_2 - \theta_1$ and θ_3, and we took $\mathbf{x}$ to be 5 equally spaced percentiles, as detailed in Appendix G.

The MDN trained with prior has two hidden layers of 50 tanh units each, whereas the MDN-SVI used to train the proposal prior and the one trained with proposal have one hidden layer of 50 tanh units. As observed in the Lotka–Volterra demo, less capacity is required when training with proposal, as the relationship to be learned is local and hence simpler, which saves compute time and gives a more accurate final posterior. All MDNs have 8 Gaussian components (except the MDN-SVI used to train the proposal prior, which always has one), which, after experimentation, we determined are enough for the MDNs to represent the non-Gaussian nature of the posterior.

Figure 4 reports the log probability of the true parameters under each posterior learnt—for ABC, the log probability was calculated by fitting a mixture of 8 Gaussians to posterior samples using Expectation-Maximization—and the number of simulations needed to achieve it. As before, MDN methods are more confident compared to ABC around the true parameters, which is due to ABC computing a broader posterior than the true one. MDN methods make more efficient use of simulations, since they use all of them for training and, unlike ABC, do not throw a proportion of them away.

4 Related work

Regression adjustment. An early parametric approach to ABC is *regression adjustment*, where a parametric regressor is trained on simulation data in order to learn a mapping from $\mathbf{x}$ to $\boldsymbol{\theta}$. The learnt mapping is then used to correct for using a large ϵ, by adjusting the location of posterior samples gathered by e.g. rejection ABC. Beaumont et al. [1] used linear regressors, and later Blum and François [4] used neural networks with one hidden layer that separately predicted the mean and variance of $\boldsymbol{\theta}$. Both can be viewed as rudimentary density estimators and as such they are a predecessor to our work. However, they were not flexible enough to accurately estimate the posterior, and they were only used within some other ABC method to allow for a larger ϵ. In our work, we make conditional density estimation flexible enough to approximate the posterior accurately.

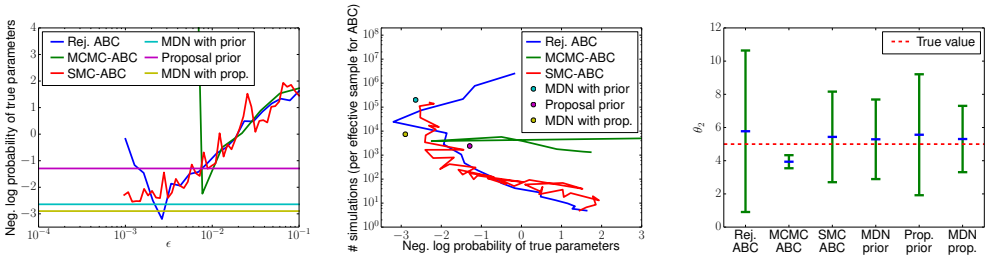


Figure 4: Results on M/G/1. **Left**: negative log probability of true parameters vs ϵ; lower is better. **Middle**: number of simulations vs negative log probability; lower left is better. Note that number of simulations is total for MDNs, and per effective sample for ABC. **Right**: Estimates of θ_2 with 2 standard deviations; ABC estimates correspond to the lowest setting of ϵ used.

Synthetic likelihood. Another parametric approach is *synthetic likelihood*, where parametric models are used to estimate the likelihood $p(\mathbf{x} \mid \boldsymbol{\theta})$. Wood [33] used a single Gaussian, and later Fan et al. [7] used a mixture Gaussian model. Both of them learnt a separate density model of $\mathbf{x}$ for each $\boldsymbol{\theta}$ by repeatedly simulating the model for fixed $\boldsymbol{\theta}$. More recently, Meeds and Welling [16] used a Gaussian process model to interpolate Gaussian likelihood approximations between different $\boldsymbol{\theta}$'s. Compared to learning the posterior, synthetic likelihood has the advantage of not depending on the choice of proposal prior. Its main disadvantage is the need of further approximate inference on top of it in order to obtain the posterior. In our work we directly learn the posterior, eliminating the need for further inference, and we address the problem of correcting for the proposal prior.

Efficient Monte Carlo ABC. Recent work on ABC has focused on reducing the simulation cost of sample-based ABC methods. Hamiltonian ABC [17] improves upon MCMC-ABC by using stochastically estimated gradients in order to explore the parameter space more efficiently. Optimization Monte Carlo ABC [18] explicitly optimizes the location of ABC samples, which greatly reduces rejection rate. Bayesian optimization ABC [11] models $p(\|\mathbf{x} - \mathbf{x}_o\| \mid \boldsymbol{\theta})$ as a Gaussian process and then uses Bayesian optimization to guide simulations towards the region of small distances $\|\mathbf{x} - \mathbf{x}_o\|$. In our work we show how a significant reduction in simulation cost can also be achieved with parametric methods, which target the posterior directly.

Recognition networks. Our use of neural density estimators for learning posteriors is reminiscent of recognition networks in machine learning. A recognition network is a neural network that is trained to invert a generative model. The Helmholtz machine [6], the variational auto-encoder [13] and stochastic backpropagation [25] are examples where a recognition network is trained jointly with the generative network it is designed to invert. Feedforward neural networks have been used to invert black-box generative models [20] and binary-valued Bayesian networks [19], and convolutional neural networks have been used to invert a physics engine [34]. Our work illustrates the potential of recognition networks in the field of likelihood-free inference, where the generative model is fixed, and inference of its parameters is the goal.

Learning proposals. Neural density estimators have been employed in learning proposal distributions for importance sampling [23] and Sequential Monte Carlo [10, 22]. Although not our focus here, our fit to the posterior could also be used within Monte Carlo inference methods. In this work we see how far we can get purely by fitting a series of conditional density estimators.

5 Conclusions

Bayesian conditional density estimation improves likelihood-free inference in three main ways: (a) it represents the posterior parametrically, as opposed to as a set of samples, allowing for probabilistic evaluations later on in the pipeline; (b) it targets the exact posterior, rather than an ϵ-approximation of it; and (c) it makes efficient use of simulations by not rejecting samples, by interpolating between samples, and by gradually focusing on the plausible parameter region. Our belief is that neural density estimation is a tool with great potential in likelihood-free inference, and our hope is that this work helps in establishing its usefulness in the field.

Acknowledgments

We thank Amos Storkey for useful comments. George Papamakarios is supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1) and the University of Edinburgh, and by Microsoft Research through its PhD Scholarship Programme.

A Proof of Proposition 1

Maximizing $\prod_n q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n)$ w.r.t. $\boldsymbol{\phi}$ is equivalent to maximizing the average log probability

$$\frac{1}{N} \sum_n \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n). \tag{6}$$

Since $(\boldsymbol{\theta}_n, \mathbf{x}_n) \sim \tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})$, due to the strong law of large numbers, as $N \to \infty$ the average log probability converges almost surely to the following expectation

$$\frac{1}{N} \sum_n \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n) \xrightarrow{\text{a.s.}} \left\langle \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) \right\rangle_{\tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})}. \tag{7}$$

Let $\tilde{p}(\mathbf{x})$ be a distribution over $\mathbf{x}$. Maximizing the above expectation w.r.t. $\boldsymbol{\phi}$ is equivalent to minimizing

$$D_{\mathrm{KL}}(\tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta}) \,\|\, \tilde{p}(\mathbf{x})\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})) = - \left\langle \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) \right\rangle_{\tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})} + \text{const.} \tag{8}$$

The above KL divergence is minimized (and becomes 0) if and only if

$$\tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta}) = \tilde{p}(\mathbf{x})\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) \tag{9}$$

almost everywhere. It is easy to see that this can only happen for $\tilde{p}(\mathbf{x}) = \int \tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}$, since

$$\tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta}) = \tilde{p}(\mathbf{x})\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) \;\Rightarrow\; \int \tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta} = \tilde{p}(\mathbf{x}) \int q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})\, \mathrm{d}\boldsymbol{\theta} = \tilde{p}(\mathbf{x}). \tag{10}$$

Thus, taking $\tilde{p}(\mathbf{x})$ as above, and assuming a setting of $\boldsymbol{\phi}$ that makes the KL equal to 0 exists, the KL is minimized if and only if we have almost everywhere that

$$q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) = \frac{\tilde{p}(\boldsymbol{\theta})}{\tilde{p}(\mathbf{x})}\, p(\mathbf{x} \,|\, \boldsymbol{\theta}) = \frac{\tilde{p}(\boldsymbol{\theta})}{\tilde{p}(\mathbf{x})}\, \frac{p(\boldsymbol{\theta} \,|\, \mathbf{x})\, p(\mathbf{x})}{p(\boldsymbol{\theta})} \propto \frac{\tilde{p}(\boldsymbol{\theta})}{p(\boldsymbol{\theta})}\, p(\boldsymbol{\theta} \,|\, \mathbf{x}). \tag{11}$$

A corollary of the above is that

$$q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) = \frac{\tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})}{\int \tilde{p}(\boldsymbol{\theta})\, p(\mathbf{x} \,|\, \boldsymbol{\theta})\, \mathrm{d}\boldsymbol{\theta}}, \tag{12}$$

in other words, $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ becomes what the posterior would be if the prior were $\tilde{p}(\boldsymbol{\theta})$.

B Parameterization and training of Mixture Density Networks

A Mixture Density Network (MDN) [3] is a conditional density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$, which takes the form of a mixture of K Gaussian components, as follows

$$q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) = \sum_k \alpha_k\, \mathcal{N}(\boldsymbol{\theta} \,|\, \mathbf{m}_k, \mathbf{S}_k). \tag{13}$$

The mixing coefficients $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_K)$, means $\{\mathbf{m}_k\}$ and covariance matrices $\{\mathbf{S}_k\}$ are computed by a feedforward neural network $f_{\mathbf{W},\mathbf{b}}(\mathbf{x})$, which has input $\mathbf{x}$, weights $\mathbf{W}$ and biases $\mathbf{b}$. In particular, let the output of the neural network be

$$\mathbf{y} = f_{\mathbf{W},\mathbf{b}}(\mathbf{x}). \tag{14}$$

Then, the mixing coefficients are given by

$$\boldsymbol{\alpha} = \mathrm{softmax}(\mathbf{W}_{\boldsymbol{\alpha}} \mathbf{y} + \mathbf{b}_{\boldsymbol{\alpha}}). \tag{15}$$

The softmax ensures that the mixing coefficients are strictly positive and sum to one. Similarly, the means are given by

$$\mathbf{m}_k = \mathbf{W}_{\mathbf{m}_k}\mathbf{y} + \mathbf{b}_{\mathbf{m}_k}. \tag{16}$$

As for the covariance matrices, we need to ensure that they are symmetric and positive definite. For this reason, instead of parameterizing the covariance matrices directly, we parameterize the Cholesky factorization of their inverses. That is, we rewrite

$$\mathbf{S}_k^{-1} = \mathbf{U}_k^T\mathbf{U}_k, \tag{17}$$

where $\mathbf{U}_k$ is parameterized to be an upper triangular matrix with strictly positive elements in the diagonal, as follows

$$\mathrm{diag}(\mathbf{U}_k) = \exp\big(\mathbf{W}_{\mathrm{diag}(\mathbf{U}_k)}\mathbf{y} + \mathbf{b}_{\mathrm{diag}(\mathbf{U}_k)}\big) \tag{18}$$

$$\mathrm{utri}(\mathbf{U}_k) = \mathbf{W}_{\mathrm{utri}(\mathbf{U}_k)}\mathbf{y} + \mathbf{b}_{\mathrm{utri}(\mathbf{U}_k)} \tag{19}$$

$$\mathrm{ltri}(\mathbf{U}_k) = \mathbf{0}. \tag{20}$$

In the above, $\mathrm{diag}(\cdot)$ picks out the diagonal elements, whereas $\mathrm{utri}(\cdot)$ and $\mathrm{ltri}(\cdot)$ pick out the elements above and below the diagonal respectively. We chose to parameterize the factorization of $\mathbf{S}_k^{-1}$ rather than that of $\mathbf{S}_k$, since it is the inverse covariance that directly appears in the calculation of $\mathcal{N}(\boldsymbol{\theta} \,|\, \mathbf{m}_k, \mathbf{S}_k)$. Apart from ensuring the symmetry and positive definiteness of $\mathbf{S}_k$, the above parameterization also allows for efficiently calculating the log determinant of $\mathbf{S}_k$ as follows

$$-\frac{1}{2}\log\det(\mathbf{S}_k) = \mathrm{sum}\big(\mathbf{W}_{\mathrm{diag}(\mathbf{U}_k)}\mathbf{y} + \mathbf{b}_{\mathrm{diag}(\mathbf{U}_k)}\big). \tag{21}$$

The above parameterization of the covariance matrix was introduced by Williams [32] for learning conditional Gaussians.

Given a set of training data $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}$, training the MDN with maximum likelihood amounts to maximizing the average log probability

$$\frac{1}{N}\sum_n \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n) \tag{22}$$

with respect to the MDN parameters

$$\boldsymbol{\phi} = \big(\mathbf{W}, \mathbf{b}, \mathbf{W}_{\boldsymbol{\alpha}}, \mathbf{b}_{\boldsymbol{\alpha}}, \big\{\mathbf{W}_{\mathbf{m}_k}, \mathbf{b}_{\mathbf{m}_k}, \mathbf{W}_{\mathrm{diag}(\mathbf{U}_k)}, \mathbf{b}_{\mathrm{diag}(\mathbf{U}_k)}, \mathbf{W}_{\mathrm{utri}(\mathbf{U}_k)}, \mathbf{b}_{\mathrm{utri}(\mathbf{U}_k)}\big\}\big). \tag{23}$$

Because the reparameterization $\boldsymbol{\phi}$ described above is unconstrained, any off-the-shelf gradient-based stochastic optimizer can be used. Gradients of the average log probability can be easily computed with backpropagation. In our experiments, we implemented MDNs using Theano [28], which automatically backpropagates gradients, and we maximized the average log likelihood using Adam [12], which is currently the state of the art in minibatch-based stochastic optimization.

C Analytical calculation of parameter posterior

According to Proposition 1, after training $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$, the posterior at $\mathbf{x} = \mathbf{x}_o$ is approximated by

$$\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) \propto \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})}\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o). \tag{24}$$

Typically, the prior $p(\boldsymbol{\theta})$ is a simple distribution like a uniform or a Gaussian. Here we will consider the uniform case, while the Gaussian case is treated analogously. Let $p(\boldsymbol{\theta})$ be uniform everywhere (improper). Then the posterior estimate becomes

$$\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) \propto \frac{q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)}{\tilde{p}(\boldsymbol{\theta})}. \tag{25}$$

In practice, we also used this estimate for uniform priors with broad but finite support. Since $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ is a mixture of K Gaussians and $\tilde{p}(\boldsymbol{\theta})$ is a single Gaussian, that is

$$q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}) = \sum_k \alpha_k\, \mathcal{N}(\boldsymbol{\theta} \,|\, \mathbf{m}_k, \mathbf{S}_k) \quad \text{and} \quad \tilde{p}(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \,|\, \mathbf{m}_0, \mathbf{S}_0), \tag{26}$$

their ratio can be calculated and normalized analytically. In particular, after some algebra it can be shown that the posterior estimate $\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ is also a mixture of K Gaussians

$$\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) = \sum_k \alpha'_k \, \mathcal{N}(\boldsymbol{\theta} \,|\, \mathbf{m}'_k, \mathbf{S}'_k), \tag{27}$$

whose parameters are

$$\mathbf{S}'_k = \left(\mathbf{S}_k^{-1} - \mathbf{S}_0^{-1}\right)^{-1} \tag{28}$$

$$\mathbf{m}'_k = \mathbf{S}'_k\left(\mathbf{S}_k^{-1}\mathbf{m}_k - \mathbf{S}_0^{-1}\mathbf{m}_0\right) \tag{29}$$

$$\alpha'_k = \frac{\alpha_k \exp\left(-\frac{1}{2}c_k\right)}{\sum_{k'} \alpha_{k'} \exp\left(-\frac{1}{2}c_{k'}\right)}, \tag{30}$$

where quantities $\{c_k\}$ are given by

$$c_k = \log\det\mathbf{S}_k - \log\det\mathbf{S}_0 - \log\det\mathbf{S}'_k + \mathbf{m}_k^T\mathbf{S}_k^{-1}\mathbf{m}_k - \mathbf{m}_0^T\mathbf{S}_0^{-1}\mathbf{m}_0 - \mathbf{m}_k'^T\mathbf{S}_k'^{-1}\mathbf{m}'_k. \tag{31}$$

For the above mixture to be well defined, the covariance matrices $\{\mathbf{S}'_k\}$ must be positive definite. This will not be the case if the proposal prior $\tilde{p}(\boldsymbol{\theta})$ is narrower than some component of $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ along some dimension. However, in both Algorithms 1 and 2, $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ is trained on parameters sampled from $\tilde{p}(\boldsymbol{\theta})$, hence, if trained properly, it tends to be narrower than $\tilde{p}(\boldsymbol{\theta})$. Our experience with Algorithms 1 and 2 is that $\{\mathbf{S}'_k\}$ not being positive definite rarely happens, whereas it happening is an indication that the algorithm's parameters have not been set up properly.

D Stochastic Variational Inference for Mixture Density Networks

In this section we describe our adaptation of Stochastic Variational Inference (SVI) for neural networks [13], in order to develop a Bayesian version of MDN. The first step is to express beliefs about the MDN parameters $\boldsymbol{\phi}$ as independent Gaussian random variables with means $\boldsymbol{\phi}_m$ and log variances $\boldsymbol{\phi}_s$. Under this interpretation we can rewrite the parameters as

$$\boldsymbol{\phi} = \boldsymbol{\phi}_m + \exp\left(\frac{1}{2}\boldsymbol{\phi}_s\right) \odot \mathbf{u}, \tag{32}$$

where the symbol $\odot$ denotes elementwise multiplication and $\mathbf{u}$ is an unknown vector drawn from a standard normal,

$$\mathbf{u} \sim \mathcal{N}(\mathbf{u} \,|\, \mathbf{0}, \mathbf{I}). \tag{33}$$

The above parameterization induces the following variational distribution over $\boldsymbol{\phi}$

$$q(\boldsymbol{\phi}) = \mathcal{N}(\boldsymbol{\phi} \,|\, \boldsymbol{\phi}_m, \mathrm{diag}(\exp\boldsymbol{\phi}_s)), \tag{34}$$

where $\mathrm{diag}(\exp\boldsymbol{\phi}_s)$ denotes a diagonal covariance matrix whose diagonal is the vector $\exp\boldsymbol{\phi}_s$. Moreover, we place the following Bayesian prior over $\boldsymbol{\phi}$

$$p(\boldsymbol{\phi}) = \mathcal{N}\left(\boldsymbol{\phi} \,|\, \mathbf{0}, \lambda^{-1}\mathbf{I}\right). \tag{35}$$

Under this prior, before seeing any data we set the parameter means $\boldsymbol{\phi}_m$ all to zero, and the parameter log variances $\boldsymbol{\phi}_s$ all equal to $\log\lambda^{-1}$. In our experiments, we used a default value of $\lambda = 0.01$.

Given training data $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}$, the objective of SVI is to optimize $\boldsymbol{\phi}_m$ and $\boldsymbol{\phi}_s$ so as to make the variational distribution $q(\boldsymbol{\phi})$ be as close as possible (in KL) to the true Bayesian posterior over $\boldsymbol{\phi}$. This objective is equivalent to maximizing a variational lower bound,

$$\frac{1}{N}\sum_n \left\langle \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n)\right\rangle_{\mathcal{N}(\mathbf{u} \,|\, \mathbf{0}, \mathbf{I})} - \frac{1}{N} D_{\mathrm{KL}}(q(\boldsymbol{\phi}) \,\|\, p(\boldsymbol{\phi})), \tag{36}$$

with respect to $\boldsymbol{\phi}_m$ and $\boldsymbol{\phi}_s$. The expectations in the first term of the above can be stochastically approximated by randomly drawing $\mathbf{u}$'s from a standard normal. The KL term can be calculated analytically, which yields

$$D_{\mathrm{KL}}(q(\boldsymbol{\phi}) \,\|\, p(\boldsymbol{\phi})) = \frac{\lambda}{2}\left(\|\boldsymbol{\phi}_m\|^2 + \mathrm{sum}(\exp\boldsymbol{\phi}_s)\right) - \frac{1}{2}\mathrm{sum}(\boldsymbol{\phi}_s) + \mathrm{const.} \tag{37}$$

The above optimization problem has been parameterized in such a way that $\boldsymbol{\phi}_m$ and $\boldsymbol{\phi}_s$ are unconstrained. Moreover, the derivatives of the variational lower bound with respect to $\boldsymbol{\phi}_m$ and $\boldsymbol{\phi}_s$ can be easily calculated with backpropagation after stochastic approximations to the expectations have been made. This allows the use of any off-the-shelf gradient-based stochastic optimizer. In our experiments, we implemented MDN-SVI in Theano [28], which automatically calculates derivatives with backpropagation, and used Adam [12] for stochastic maximization of the variational lower bound.

An important practical detail for stochastically approximating the expectation terms is the *local reparameterization trick* [14], which leverages the layered feedforward structure of the MDN. Consider any hidden or output unit in the MDN; if a is its activation and $\mathbf{z}$ is the vector of its inputs, then the relationship between a and $\mathbf{z}$ is always of the form

$$a = \mathbf{w}^T \mathbf{z} + b, \tag{38}$$

where $\mathbf{w}$ and b are the weights and bias respectively associated with this unit. As we have seen, in the SVI framework these weights and biases are Gaussian random variables with means $\mathbf{w}_m$ and b_m, and log variances $\mathbf{w}_s$ and b_s. It is easy to see that this induces a Gaussian distribution over activation a, whose mean a_m and variance $\exp a_s$ is given by

$$a_m = \mathbf{w}_m^T \mathbf{z} + b_m \quad \text{and} \quad \exp a_s = (\exp \mathbf{w}_s)^T (\mathbf{z} \odot \mathbf{z}) + \exp b_s, \tag{39}$$

where $\odot$ denotes elementwise multiplication. Therefore, randomly sampling $\mathbf{w}$ and b in order to estimate the expectations and their gradients in the variational lower bound is equivalent to directly sampling a from a Gaussian with the above mean and variance. This trick saves computation by reducing calls to the random number generator, but more importantly it reduces the variance of the stochastic approximation of the expectations (intuitively this is because less randomness is involved) and hence it makes stochastic optimization more stable and faster to converge.

E Effective sample size of ABC methods

Rejection ABC returns a set of *independent* samples, MCMC-ABC returns a set of *correlated* samples, and SMC-ABC returns a set of independent but *weighted* samples. To make a fair comparison between them in terms of simulation cost, we quote the number of simulations per *effective* sample, that is, the total number of simulations divided by the effective sample size of the returned set of samples.

Let $\{\boldsymbol{\theta}_n\}$ be a set of N samples, not necessarily independent. The effective sample size N_{eff} is defined to be the number of equivalent *independent* samples that would give an estimator of equal variance. For rejection ABC $N_{\text{eff}} = N$, since all returned samples are independent.

Suppose that each sample is a vector of D components. For MCMC-ABC, where samples come in the form of D autocorrelated sequences, we estimated the effective sample size for component d as

$$N_{\text{eff},d} = \frac{N}{1 + 2\sum_{l=1}^{L_d} r_{dl}}, \tag{40}$$

where r_{dl} is the autocorrelation coefficient of component d at lag l, estimated from the samples. We calculated the summation up to lag L_d, which corresponds to the first autocorrelation coefficient that is equal to 0. Then we took the effective sample size N_{eff} to be the minimum $N_{\text{eff},d}$ across components. For a more general discussion on estimating autocorrelation time (which is equal to N/N_{eff} and thus equivalent to effective sample size) see Thompson [29].

For SMC-ABC, each sample is independent but comes with a corresponding non-negative weight w_n. The weights have to sum to one, that is $\sum_n w_n = 1$. We estimated the effective sample size by

$$N_{\text{eff}} = \frac{1}{\sum_n w_n^2}. \tag{41}$$

It is easy to see that if $w_n = 1/N$ for all n then $N_{\text{eff}} = N$, and if all weights but one are 0 then $N_{\text{eff}} = 1$. For a discussion regarding the above estimate see Nowozin [21].

F Setup for the Lotka–Volterra experiment

The Lotka–Volterra model [30] is a stochastic model that was developed to describe the time evolution of a population of predators interacting with a population of prey. Let X be the number of predators

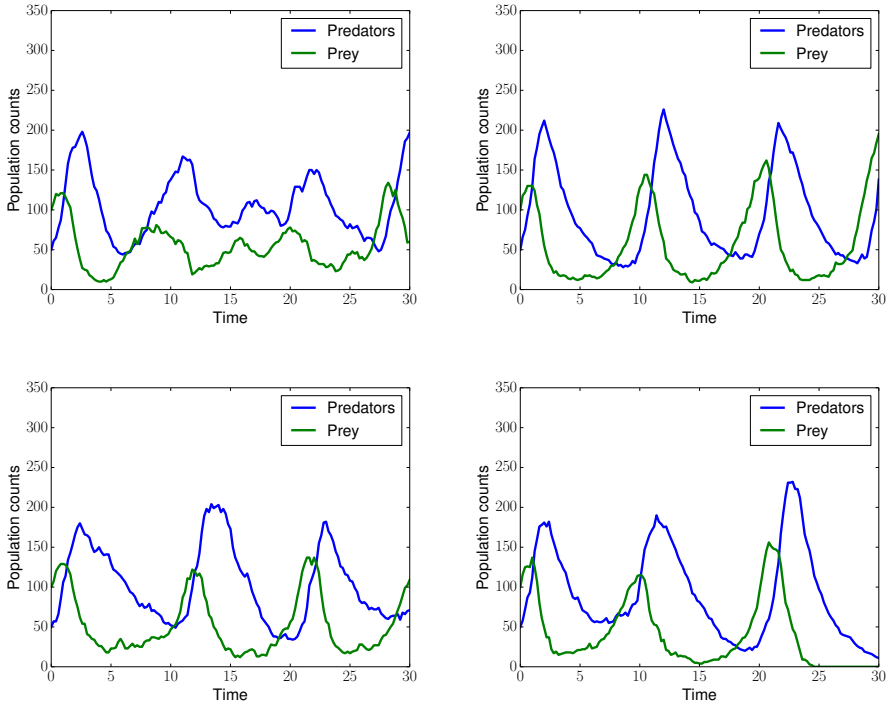


Figure 5: Typical oscillatory behaviour of predator/prey populations corresponding to four different simulations of the Lotka–Volterra model with parameter values $\theta_1 = 0.01$, $\theta_2 = 0.5$, $\theta_3 = 1$, and $\theta_4 = 0.01$.

and Y be the number of prey. The model asserts that the following four reactions can take place, with corresponding rates:

(i) A predator may be born, with rate $\theta_1 XY$, increasing X by one.

(ii) A predator may die, with rate $\theta_2 X$, decreasing X by one.

(iii) A prey may be born, with rate $\theta_3 Y$, increasing Y by one.

(iv) A prey may be eaten by a predator, with rate $\theta_4 XY$, decreasing Y by one.

Given initial populations X and Y, the above model can be simulated using Gillespie's algorithm [8], as follows:

(i) Draw the time to next reaction from an exponential distribution with rate equal to the total rate $\theta_1 XY + \theta_2 X + \theta_3 Y + \theta_4 XY$.

(ii) Select a reaction at random, with probability proportional to its rate.

(iii) Simulate the reaction, and go to step (i).

In our experiments, each simulation started with initial populations $X = 50$ and $Y = 100$, and took place for a total of 30 time units. We recorded the values of X and Y after every 0.2 time units, resulting in two time series of 151 values each.

Data $\mathbf{x}$ was taken to be the following set of 9 statistics calculated from the time series:

(i) The mean of each time series.

(ii) The log variance of each time series.

(iii) The autocorrelation coefficient of each time series at lag 1 and lag 2.

(iv) The cross-correlation coefficient between the two time series.

Since the above statistics have potentially very different scales, we normalized them on the basis of a pilot run. That is, we performed a pilot run of 1000 simulations, calculated and stored the mean and standard deviation of each statistic across pilot simulations, and used them in all subsequent simulations to normalize each statistic by subtracting the pilot mean and dividing by the pilot standard deviation. This choice of statistics and normalization process was taken from Wilkinson [31].

From our experience with the model we observed that typical evolutions of the predator/prey populations for randomly selected parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3, \theta_4)$ include (a) the predators quickly eating all the prey and then slowly decaying exponentially, or (b) the predators quickly dying out and then the prey growing exponentially. However, for certain carefully tuned values of $\boldsymbol{\theta}$, the two populations exhibit an oscillatory behaviour, typical of natural ecological systems. In order to generate observations $\mathbf{x}_o$ for our experimental setup, we set the parameters to

$$\theta_1 = 0.01, \quad \theta_2 = 0.5, \quad \theta_3 = 1, \quad \theta_4 = 0.01 \tag{42}$$

and simulated the model to generate $\mathbf{x}_o$. We carefully chose parameter values that give rise to oscillatory behaviour (see Figure 5 for typical examples of population evolution corresponding to the above parameters). Since only a small subset of parameters give rise to such oscillatory behaviour, the posterior $p(\boldsymbol{\theta} \mid \mathbf{x} = \mathbf{x}_o)$ is expected to be tightly peaked around the true parameter values. We tested our algorithms by evaluating how well (in terms of assigned log probability) each algorithm retrieves the true parameters.

Finally, we took the prior over $\boldsymbol{\theta}$ to be uniform in the log domain. That is, the prior was taken to be

$$p(\log \boldsymbol{\theta}) \propto \prod_{i=1}^{4} \mathcal{U}(\log \theta_i \mid \log \theta_\alpha, \log \theta_\beta), \tag{43}$$

where $\log \theta_\alpha = -5$ and $\log \theta_\beta = 2$, which of course includes the true parameters. All our inferences where done in the log domain.

G Setup for the M/G/1 experiment

The M/G/1 queue model [27] is a statistical model that describes how a single server processes a queue formed by a set of continuously arriving jobs. Let I be the total number of jobs to be processed, s_i be the time the server takes to process job i, v_i be the time that job i entered the queue, and d_i be the time that job i left the queue (i.e. the time when the server finished processing it). The M/G/1 queue model asserts that for each job i we have

$$s_i \sim \mathcal{U}(\theta_1, \theta_2) \tag{44}$$

$$v_i - v_{i-1} \sim \mathrm{Exp}(\theta_3) \tag{45}$$

$$d_i - d_{i-1} = s_i + \max(0, v_i - d_{i-1}). \tag{46}$$

In the above equations, $\mathcal{U}(\theta_1, \theta_2)$ denotes a uniform distribution in the range $[\theta_1, \theta_2]$, $\mathrm{Exp}(\theta_3)$ denotes an exponential distribution with rate θ_3, and $v_0 = d_0 = 0$. In our experiments we used a total of $I = 50$ jobs.

The goal is to infer parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$ if the only knowledge is a set of percentiles of the empirical distribution of the interdeparture times $d_i - d_{i-1}$ for $i = 1, \ldots, I$. In our experiments we used 5 equally spaced percentiles. That is, given a set of I interdeparture times $d_i - d_{i-1}$, we took $\mathbf{x}$ to be the 0th, 25th, 50th, 75th and 100th percentiles of the set of interdeparture times. Note that the 0th and 100th percentiles correspond to the minimum and maximum element in the set.

Since different percentiles can have different scales and strong correlations between them, we whitened the data on the basis of a pilot run. That is, we performed 100K pilot simulations, and recorded the mean vector and covariance matrix of the resulting percentiles. For each subsequent simulation, we calculated $\mathbf{x}$ from resulting percentiles by subtracting the mean vector and decorrelating and normalizing with the covariance matrix.

To generate observed data $\mathbf{x}_o$, we set the parameters to the following values

$$\theta_1 = 1, \quad \theta_2 = 5, \quad \theta_3 = 0.2 \tag{47}$$

and simulated the model to get $\mathbf{x}_o$. We evaluated inference algorithms by how well the true parameter values were retrieved, as measured by log probability under computed posteriors. Finally, the prior

probability of the parameters was taken to be

$$\theta_1 \sim \mathcal{U}(0, 10) \tag{48}$$

$$\theta_2 - \theta_1 \sim \mathcal{U}(0, 10) \tag{49}$$

$$\theta_3 \sim \mathcal{U}(0, 1/3), \tag{50}$$

which is uniform, albeit not axis-aligned, and of course includes the true parameters.

References

[1] M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate Bayesian Computation in population genetics. *Genetics*, 162:2025–2035, Dec. 2002.

[2] M. A. Beaumont, J.-M. Cornuet, J.-M. Marin, and C. P. Robert. Adaptive Approximate Bayesian Computation. *Biometrika*, 96(4):983–990, 2009.

[3] C. M. Bishop. Mixture density networks. Technical Report NCRG/94/004, Aston University, 1994.

[4] M. G. B. Blum and O. François. Non-linear regression models for Approximate Bayesian Computation. *Statistics and Computing*, 20(1):63–73, 2010.

[5] F. V. Bonassi and M. West. Sequential Monte Carlo with adaptive weights for Approximate Bayesian Computation. *Bayesian Analysis*, 10(1):171–187, Mar. 2015.

[6] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The Helmholtz machine. *Neural Computation*, 7: 889–904, 1995.

[7] Y. Fan, D. J. Nott, and S. A. Sisson. Approximate Bayesian Computation via regression density estimation. *Stat*, 2(1):34–48, 2013.

[8] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[9] C. Gouriéroux, A. Monfort, and E. Renault. Indirect inference. *Journal of Applied Econometrics*, 8(S1): S85–S118, 1993.

[10] S. Gu, Z. Ghahramani, and R. E. Turner. Neural adaptive Sequential Monte Carlo. *Advances in Neural Information Processing Systems 28*, pages 2629–2637, 2015.

[11] M. U. Gutmann and J. Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *arXiv e-prints*, abs/1501.03291v3, 2015.

[12] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations*, 2014.

[13] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*, 2013.

[14] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *Advances in Neural Information Processing Systems 28*, pages 2575–2583, 2015.

[15] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, Dec. 2003.

[16] E. Meeds and M. Welling. GPS-ABC: Gaussian Process Surrogate Approximate Bayesian Computation. *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, 30, 2014.

[17] E. Meeds, R. Leenders, and M. Welling. Hamiltonian ABC. *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, pages 582–591, 2015.

[18] T. Meeds and M. Welling. Optimization Monte Carlo: Efficient and embarrassingly parallel likelihood-free inference. *Advances in Neural Information Processing Systems 28*, pages 2071–2079, 2015.

[19] Q. Morris. Recognition networks for approximate inference in BN20 networks. *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, pages 370–377, 2001.

[20] V. Nair, J. Susskind, and G. E. Hinton. Analysis-by-synthesis by learning to invert generative black boxes. *Proceedings of the 18th International Conference on Artificial Neural Networks*, 5163:971–981, 2008.

[21] S. Nowozin. Effective sample size in importance sampling, Aug. 2015. URL `http://www.nowozin.net/sebastian/blog/effective-sample-size-in-importance-sampling.html`. Accessed on 18 May 2016.

[22] B. Paige and F. Wood. Inference networks for Sequential Monte Carlo in graphical models. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[23] G. Papamakarios and I. Murray. Distilling intractable generative models. *Probabilistic Integration Workshop at Neural Information Processing Systems*, 2015.

[24] J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman. Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular Biology and Evolution*, 16(12): 1791–1798, 1999.

[25] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *Proceedings of the 31st International Conference on Machine Learning*, pages 1278–1286, 2014.

[26] C. M. Schafer and P. E. Freeman. Likelihood-free inference in cosmology: Potential for the estimation of luminosity functions. *Statistical Challenges in Modern Astronomy V*, pages 3–19, 2012.

[27] A. Y. Shestopaloff and R. M. Neal. On Bayesian inference for the M/G/1 queue with efficient MCMC sampling. *arXiv e-prints*, abs/1401.5548v1, 2014.

[28] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688v1, May 2016.

[29] M. B. Thompson. A comparison of methods for computing autocorrelation time. *arXiv e-prints*, abs/1011.0175v1, Oct. 2010.

[30] D. J. Wilkinson. *Stochastic Modelling for Systems Biology, Second Edition*. Chapman & Hall/CRC Mathematical and Computational Biology. Taylor & Francis, 2011.

[31] D. J. Wilkinson. Summary stats for ABC, Sept. 2013. URL `https://darrenjw.wordpress.com/2013/09/01/summary-stats-for-abc/`. Accessed on 16 May 2016.

[32] P. M. Williams. Using neural networks to model conditional multivariate densities. *Neural Computation*, 8 (4):843–854, May 1996.

[33] S. N. Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466(7310): 1102–1104, 2010.

[34] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. *Advances in Neural Information Processing Systems 28*, pages 127–135, 2015.

3.4 Discussion

I conclude this chapter with a discussion of the paper *Fast ϵ-free Inference of Simulator Models with Bayesian Conditional Density Estimation*, which was presented in the previous section. In what follows, I will discuss the paper's contribution and impact so far, I will compare the proposed method with regression adjustment, I will criticize the paper's limitations, and I will discuss further advances inspired by the proposed method.

3.4.1 Contribution and impact

The paper establishes neural density estimation as a viable method for likelihood-free inference. Inference is viewed as a density-estimation problem, where the unknown posterior is the target density to estimate, and the simulator is the source of training data. Unlike ABC methods such as those discussed in section 3.2, likelihood-free inference via neural density estimation doesn't involve accepting/rejecting parameter samples, and consequently it doesn't require specifying a tolerance ϵ. The paper demonstrates that it is possible for neural density models to estimate joint posterior densities accurately, and without the need to replace the exact posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ by an approximate posterior $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$ as typically done in ABC.

Furthermore, the paper demonstrates that likelihood-free inference via neural density estimation can lead to massive computational savings in simulation cost compared to traditional ABC. This is achieved by a sequential training procedure, where preliminary estimates of the posterior are used as proposals for obtaining more training data. This sequential procedure is similar to the one used in sequential Monte Carlo ABC, where new parameters are proposed by perturbing previous posterior samples obtained with a larger ϵ (as explained in section 3.2.3). As the paper shows, the proposed method can estimate the joint density of the entire posterior with roughly the same number of simulations as SMC-ABC needs to produce (the equivalent of) a single independent posterior sample.

According to Google Scholar, the paper has received 41 citations as of April 2019. The proposed method has directly influenced subsequent developments in likelihood-free inference, such as *Sequential Neural Posterior Estimation* (Lueckmann et al., 2017), which will be discussed in more detail in section 3.4.4, *Sequential Neural Likelihood* (Papamakarios et al., 2019), which is the topic of next chapter, and *adaptive Gaussian-copula ABC* (Chen and Gutmann, 2019). Finally, variants of the proposed method have been used in applications of likelihood-free inference in cosmology (Alsing et al., 2018a, 2019) and in neuroscience (Lueckmann et al., 2017).

3.4.2 Comparison with regression adjustment

As mentioned in section 4 of the paper, *regression adjustment* is closely related to our work, and can be seen as a predecessor to likelihood-free inference via neural density estimation. Here, I discuss in more depth what regression adjustment is, in what way it is similar to neural density estimation, and also how it differs.

Regression adjustment is typically performed as a post-processing step after rejection ABC, to correct for the approximation incurred by using a non-zero tolerance ϵ. Suppose we run rejection ABC with some tolerance ϵ, until we obtain a set of N joint samples $\{(\boldsymbol{\theta}_1, \mathbf{x}_1), \ldots, (\boldsymbol{\theta}_N, \mathbf{x}_N)\}$. As we have seen, each pair $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ is an independent joint sample from the following distribution:

$$p(\boldsymbol{\theta}, \mathbf{x} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon) \propto I(\|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)\, p(\mathbf{x} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta}). \tag{3.38}$$

Marginally, each $\boldsymbol{\theta}_n$ is an independent sample from the approximate posterior $p(\boldsymbol{\theta} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$. If $\epsilon = \infty$ then all simulations are accepted, in which case each $\boldsymbol{\theta}_n$ is an independent sample from

the prior $p(\boldsymbol{\theta})$. The goal of regression adjustment is to transform each sample $\boldsymbol{\theta}_n$ into an 'adjusted' sample $\boldsymbol{\theta}'_n$, such that $\boldsymbol{\theta}'_n$ is an independent sample from the exact posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$.

The first step of regression adjustment is to estimate the stochastic mapping from $\mathbf{x}$ to $\boldsymbol{\theta}$, where $\mathbf{x}$ and $\boldsymbol{\theta}$ are obtained by rejection ABC and are jointly distributed according to $p(\boldsymbol{\theta}, \mathbf{x} \,|\, \|\mathbf{x} - \mathbf{x}_o\| \leq \epsilon)$ as described above. Regression adjustment models this mapping with a parametric regressor of the following form:

$$\boldsymbol{\theta} = f_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{x}), \tag{3.39}$$

where $\mathbf{u}$ is random noise from some distribution $\pi_u(\mathbf{u})$ that doesn't depend on $\mathbf{x}$, and $\boldsymbol{\phi}$ are the parameters of the regressor. The random noise $\mathbf{u}$ accounts for the stochasticity of the mapping from $\mathbf{x}$ to $\boldsymbol{\theta}$. We don't need to specify the noise distribution $\pi_u(\mathbf{u})$, but we need to design the function $f_{\boldsymbol{\phi}}(\cdot, \mathbf{x})$ to be invertible, so that given $\boldsymbol{\theta}$ and $\mathbf{x}$ we can easily solve for $\mathbf{u}$. For example, Beaumont et al. (2002) used the following linear regressor:

$$\boldsymbol{\theta} = \mathbf{A}\mathbf{x} + \boldsymbol{\beta} + \mathbf{u} \quad \text{with} \quad \boldsymbol{\phi} = \{\mathbf{A}, \boldsymbol{\beta}\}, \tag{3.40}$$

whereas Blum and François (2010) used the following non-linear regressor:

$$\boldsymbol{\theta} = g_{\boldsymbol{\phi}_1}(\mathbf{x}) + h_{\boldsymbol{\phi}_2}(\mathbf{x}) \odot \mathbf{u} \quad \text{with} \quad \boldsymbol{\phi} = \{\boldsymbol{\phi}_1, \boldsymbol{\phi}_2\}. \tag{3.41}$$

In the above, $g_{\boldsymbol{\phi}_1}$ and $h_{\boldsymbol{\phi}_2}$ are neural networks parameterized by $\boldsymbol{\phi}_1$ and $\boldsymbol{\phi}_2$, and $\odot$ denotes the elementwise product.

In practice, the regressor $f_{\boldsymbol{\phi}}$ is trained on the set of samples $\{(\boldsymbol{\theta}_1, \mathbf{x}_1), \ldots, (\boldsymbol{\theta}_N, \mathbf{x}_N)\}$ obtained by rejection ABC. Assuming the noise $\mathbf{u}$ has zero mean, the linear regressor in equation (3.40) can be trained by minimizing the average squared error on the parameters as follows:

$$(\mathbf{A}^*, \boldsymbol{\beta}^*) = \underset{\mathbf{A}, \boldsymbol{\beta}}{\arg\min} \, \frac{1}{N} \sum_n \|\boldsymbol{\theta}_n - \mathbf{A}\mathbf{x}_n - \boldsymbol{\beta}\|^2. \tag{3.42}$$

The above optimization problem has a unique solution in closed form. The non-linear regressor in equation (3.41) can be trained in two stages. In the first stage, we assume that $\mathbf{u}$ has zero mean, and we fit $\boldsymbol{\phi}_1$ by minimizing the average squared error on the parameters:

$$\boldsymbol{\phi}_1^* = \underset{\boldsymbol{\phi}_1}{\arg\min} \, \frac{1}{N} \sum_n \|\boldsymbol{\theta}_n - g_{\boldsymbol{\phi}_1}(\mathbf{x}_n)\|^2. \tag{3.43}$$

In the second stage, we assume that $\log|\mathbf{u}|$ has zero mean, and we fit $\boldsymbol{\phi}_2$ by minimizing the average squared error on the log absolute residuals:

$$\boldsymbol{\phi}_2^* = \underset{\boldsymbol{\phi}_2}{\arg\min} \, \frac{1}{N} \sum_n \left\| \log\left|\boldsymbol{\theta}_n - g_{\boldsymbol{\phi}_1^*}(\mathbf{x}_n)\right| - \log\left|h_{\boldsymbol{\phi}_2}(\mathbf{x}_n)\right| \right\|^2, \tag{3.44}$$

where the log and absolute-value operators are meant to be understood elementwise. The above two optimization problems don't have a known closed-form solution, but can be approximately solved by gradient methods.

The second step of regression adjustment is to use the trained regressor $f_{\boldsymbol{\phi}^*}$ to adjust the parameter samples $\{\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_N\}$ obtained by rejection ABC. First, we can obtain N independent samples $\{\mathbf{u}_1, \ldots, \mathbf{u}_N\}$ from $\pi_u(\mathbf{u})$ by solving $\boldsymbol{\theta}_n = f_{\boldsymbol{\phi}^*}(\mathbf{u}_n, \mathbf{x}_n)$ for $\mathbf{u}_n$ separately for each n. Then, assuming $f_{\boldsymbol{\phi}^*}$ captures the exact stochastic mapping from $\mathbf{x}$ to $\boldsymbol{\theta}$, we can generate N samples $\{\boldsymbol{\theta}'_1, \ldots, \boldsymbol{\theta}'_N\}$ from the exact posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$ by simply plugging $\mathbf{u}_n$ and $\mathbf{x}_o$ back into the regressor as follows:

$$\boldsymbol{\theta}'_n = f_{\boldsymbol{\phi}^*}(\mathbf{u}_n, \mathbf{x}_o). \tag{3.45}$$

For example, using the linear regressor in equation (3.40) we obtain:

$$\boldsymbol{\theta}'_n = \mathbf{A}^* \left(\mathbf{x}_o - \mathbf{x}_n\right) + \boldsymbol{\theta}_n, \tag{3.46}$$

whereas using the non-linear regressor in equation (3.41) we obtain:

$$\boldsymbol{\theta}'_n = g_{\boldsymbol{\phi}_1^*}(\mathbf{x}_o) + h_{\boldsymbol{\phi}_2^*}(\mathbf{x}_o) \odot \frac{\boldsymbol{\theta}_n - g_{\boldsymbol{\phi}_1^*}(\mathbf{x}_n)}{h_{\boldsymbol{\phi}_2^*}(\mathbf{x}_n)}, \tag{3.47}$$

where division is meant to be understood elementwise.

In practice, $f_{\boldsymbol{\phi}^*}$ will be an approximation to the exact stochastic mapping from $\mathbf{x}$ to $\boldsymbol{\theta}$, hence the adjusted samples $\{\boldsymbol{\theta}'_1, \ldots, \boldsymbol{\theta}'_N\}$ will only approximately follow the exact posterior. For the adjusted samples to be a good description of the exact posterior, $f_{\boldsymbol{\phi}^*}$ needs to be a good model of the stochastic mapping from $\mathbf{x}$ to $\boldsymbol{\theta}$ at least within a distance ϵ away from $\mathbf{x}_o$. Consequently, a less flexible regressor (such as the linear regressor described above) requires a smaller ϵ, whereas a more flexible regressor can be used with a larger ϵ.

Similar to regression adjustment, likelihood-free inference via neural density estimation is based on modelling the stochastic relationship between $\boldsymbol{\theta}$ and $\mathbf{x}$. The difference is in the type of model employed and in the way the model is used. Neural density estimation uses a conditional-density model $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ (which in the paper is taken to be a mixture-density network) with the following characteristics:

(i) We can calculate the conditional density under $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ for any $\boldsymbol{\theta}$ and $\mathbf{x}$.

(ii) We can obtain exact independent samples from $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ for any $\mathbf{x}$.

The first property allows us to train $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ by maximum likelihood on simulated data, and thus estimate the density of the posterior. The second property allows us to obtain as many samples from the estimated posterior as we wish.

In contrast, regression adjustment models the stochastic relationship between $\boldsymbol{\theta}$ and $\mathbf{x}$ as follows:

$$\boldsymbol{\theta} = f_{\boldsymbol{\phi}}(\mathbf{u}, \mathbf{x}) \quad \text{where} \quad \mathbf{u} \sim \pi_u(\mathbf{u}). \tag{3.48}$$

We can think of the above as a reparameterized version of a neural density estimator in terms of its internal random numbers (the chapter on Masked Autoregressive Flow explores this idea in depth). The crucial difference is that regression adjustment only requires fitting the regressor $f_{\boldsymbol{\phi}}$, and it entirely avoids modelling the noise distribution $\pi_u(\mathbf{u})$. As a consequence, regression adjustment can neither calculate densities under the model, nor generate samples from the model at will, as these operations require knowing $\pi_u(\mathbf{u})$.

In principle, we can use a neural density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ to perform regression adjustment, as long as we can write $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ as an invertible transformation of noise as in equation (3.48). A mixture-density network cannot be easily expressed this way, but a model such as Masked Autoregressive Flow (or any other normalizing flow) is precisely an invertible transformation of noise by design. Hence, we can use a normalizing flow to estimate the conditional density of $\boldsymbol{\theta}$ given $\mathbf{x}$, and then obtain posterior samples by adjusting simulated data. More precisely, given a simulated pair $(\boldsymbol{\theta}_n, \mathbf{x}_n)$, we can use the trained normalizing flow to compute the noise $\mathbf{u}_n$ that would have generated $\boldsymbol{\theta}_n$ given $\mathbf{x}_n$, and then run the flow with noise $\mathbf{u}_n$ but this time conditioned on $\mathbf{x}_o$ to obtain a sample $\boldsymbol{\theta}'_n$ from the exact posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o)$.

Nevertheless, I would argue that using a neural density estimator such as a normalizing flow for regression adjustment is an unnecessarily roundabout way of generating posterior samples. If we assume that the flow is a good model of the stochastic relationship between $\boldsymbol{\theta}$ and $\mathbf{x}$ (which is what regression adjustment requires), then we may as well obtain posterior samples by sampling the noise directly from $\pi_u(\mathbf{u})$ (recall that the flow explicitly models the noise distribution). In

other words, with neural density estimation there is no need to adjust simulated data to obtain posterior samples; we can obtain as many posterior samples as we like by sampling from the neural density model directly.

3.4.3 Limitations and criticism

So far I've discussed the contributions of the paper and the impact the paper has had in the field of likelihood-free inference. In this section, I examine the paper from a critical perspective: I discuss the limitations of the proposed method, and identify its weaknesses.

Recall that the proposed method estimates the posterior in a series of rounds. In each round, parameter samples are proposed from a distribution $\tilde{p}(\boldsymbol{\theta})$, then the neural density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ is trained on simulated data, and finally the posterior is estimated as follows:

$$\hat{p}(\boldsymbol{\theta} \,|\, \mathbf{x} = \mathbf{x}_o) \propto \frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})}\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o). \tag{3.49}$$

The above posterior estimate is then used as the proposal for the next round, and the algorithm can thus progress for any number of rounds. In the first round, we may use the prior as proposal.

In order to ensure that the computation of the posterior estimate in equation (3.49) is analytically tractable, the proposed method restricts $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ to be a conditional Gaussian mixture model (i.e. a mixture-density network), and assumes that the prior $p(\boldsymbol{\theta})$ and the proposal $\tilde{p}(\boldsymbol{\theta})$ are also Gaussian. The Gaussian assumption can be relaxed to some extent; equation (3.49) remains analytically tractable if we replace Gaussians with any other exponential family. However, the proposed algorithm is still tied to mixture-density networks and can't employ more advanced neural density estimators, such as Masked Autoregressive Flow. In addition, the algorithm can't be used with arbitrary priors, which limits its applicability.

Even though mixture-density networks can be made arbitrarily flexible, I would argue that, in principle, tying an inference algorithm to a particular neural density estimator should be viewed as a weakness. Having the inference algorithm and the neural density model as separate components of an inference framework encourages modularity and good software design. Furthermore, if an inference algorithm can interface with any neural density model, improvements in neural density estimation (such as those described in the previous chapter) will automatically translate to improvements in the inference algorithm.

An important limitation of the proposed method is that equation (3.49) is not guaranteed to yield a posterior estimate that integrates to 1. For example, assume that $p(\boldsymbol{\theta}) \propto 1$, that $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ has a Gaussian component with covariance $\mathbf{S}$, and that $\tilde{p}(\boldsymbol{\theta})$ is a Gaussian with covariance $\alpha\mathbf{S}$ for some $\alpha < 1$; in other words, the proposal is narrower than one of the components of $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$. In this case, dividing $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ by the proposal yields a Gaussian component with covariance:

$$\left(\mathbf{S}^{-1} - (\alpha\mathbf{S})^{-1}\right)^{-1} = \frac{\alpha}{\alpha - 1}\,\mathbf{S}, \tag{3.50}$$

which is negative-definite. When this happens, the algorithm has no good way of continuing to make progress; in practice it terminates early and returns the posterior estimate from the previous round. The problem of negative-definite covariances resulting from dividing one Gaussian by another is also encountered in *expectation propagation* (Minka, 2001).

In appendix C, the paper states the following:

> [The neural density estimator] $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ is trained on parameters sampled from $\tilde{p}(\boldsymbol{\theta})$, hence, if trained properly, it tends to be narrower than $\tilde{p}(\boldsymbol{\theta})$.

While in practice this is often the case, I should emphasize that there is no guarantee that $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ will always be narrower than the proposal. Furthermore, the paper proceeds to state the following:

> [The covariance] not being positive-definite rarely happens, whereas it happening is an indication that the algorithm's parameters have not been set up properly.

This was indeed my experience in the experiments of the paper. However, as I continued to use the algorithm after the publication of the paper, I found that sometimes negative-definite covariances are hard to avoid, regardless of how the algorithm's parameters are set up. In fact, in the next chapter I give two examples of inference tasks in which the algorithm fails after its second round.

Since the publication of the paper, the above two issues, i.e. restricting the type of neural density estimator and terminating due to an improper posterior estimate, have motivated further research that aims to improve upon the proposed algorithm. In section 3.4.4 and in the next chapter, I will discuss methods that were developed to address these two issues.

3.4.4 Sequential Neural Posterior Estimation

About a year after the publication of the paper, the proposed method was extended by Lueckmann et al. (2017), partly in order to address the limitations described in the previous section. Lueckmann et al. (2017) called their method *Sequential Neural Posterior Estimation*. I thought that *Sequential Neural Posterior Estimation* was an excellent name indeed, and that it would be an appropriate name for our method as well, which until then had remained nameless. With Jan-Matthis Lueckmann's permission, I now use the term SNPE to refer to both methods; if I need to distinguish between them, I use SNPE-A for our method and SNPE-B for the method of Lueckmann et al. (2017).

Similar to SNPE-A, SNPE-B estimates the posterior density by training a neural density estimator over multiple rounds, where the posterior estimate of each round becomes the proposal for the next round. The main difference between SNPE-A and SNPE-B is the strategy employed in order to correct for sampling parameters from the proposal $\tilde{p}(\boldsymbol{\theta})$ instead of the prior $p(\boldsymbol{\theta})$. Recall that SNPE-A trains the neural density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ on samples from the proposal, and then analytically adjusts $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ by multiplying with $p(\boldsymbol{\theta})$ and dividing by $\tilde{p}(\boldsymbol{\theta})$, as shown in equation (3.49). As we've seen, this strategy restricts the form of the neural density estimator, and may lead to posterior estimates that don't integrate to 1.

In contrast, SNPE-B assigns importance weights to the proposed samples, and then trains the neural density estimator on the weighted samples. Let $\{(\boldsymbol{\theta}_1, \mathbf{x}_1), \ldots, (\boldsymbol{\theta}_N, \mathbf{x}_N)\}$ be a set of N independent joint samples obtained by:

$$\boldsymbol{\theta}_n \sim \tilde{p}(\boldsymbol{\theta}) \quad \text{and} \quad \mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}_n). \tag{3.51}$$

Assuming $\tilde{p}(\boldsymbol{\theta})$ is non-zero in the support of the prior, SNPE-B assigns an importance weight w_n to each joint sample $(\boldsymbol{\theta}_n, \mathbf{x}_n)$, given by:

$$w_n = \frac{p(\boldsymbol{\theta}_n)}{\tilde{p}(\boldsymbol{\theta}_n)}. \tag{3.52}$$

Then, the neural density estimator is trained by maximizing the average log likelihood on the weighted dataset, given by:

$$L(\boldsymbol{\phi}) = \frac{1}{N} \sum_n w_n \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n). \tag{3.53}$$

As $N \to \infty$, due to the strong law of large numbers, $L(\boldsymbol{\phi})$ converges almost surely to the following expectation:

$$L(\boldsymbol{\phi}) \xrightarrow{a.s.} \mathbb{E}_{\tilde{p}(\boldsymbol{\theta})p(\mathbf{x}\,|\,\boldsymbol{\theta})}\left(\frac{p(\boldsymbol{\theta})}{\tilde{p}(\boldsymbol{\theta})}\log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}\,|\,\mathbf{x})\right) = \mathbb{E}_{p(\boldsymbol{\theta},\mathbf{x})}(\log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}\,|\,\mathbf{x})). \tag{3.54}$$

Maximizing the above expectation is equivalent to minimizing $D_{\mathrm{KL}}(p(\boldsymbol{\theta},\mathbf{x}) \,\|\, q_{\boldsymbol{\phi}}(\boldsymbol{\theta}\,|\,\mathbf{x})\,p(\mathbf{x}))$, which happens only if $q_{\boldsymbol{\phi}}(\boldsymbol{\theta}\,|\,\mathbf{x}) = p(\boldsymbol{\theta}\,|\,\mathbf{x})$ almost everywhere. Therefore, the training procedure of SNPE-B targets the exact posterior (at least asymptotically), which means that we can estimate the exact posterior simply by:

$$\hat{p}(\boldsymbol{\theta}\,|\,\mathbf{x} = \mathbf{x}_o) = q_{\boldsymbol{\phi}}(\boldsymbol{\theta}\,|\,\mathbf{x}_o). \tag{3.55}$$

Unlike SNPE-A where the neural density estimator must be a mixture-density network and the prior and the proposal must be Gaussian, SNPE-B imposes no restrictions in the form of the neural density estimator, the proposal or the prior. This means that the proposal doesn't need to be Gaussian, but can be an arbitrary density model. Moreover, since the neural density estimator targets the posterior directly, we are guaranteed that the posterior estimate is a proper density, and the algorithm can proceed for any number of rounds without the risk of terminating early.

Nonetheless, SNPE-B has its own weaknesses. If the proposal $\tilde{p}(\boldsymbol{\theta})$ is not a good match to the prior $p(\boldsymbol{\theta})$ (which is likely in later rounds), the importance weights will have high variance, which means that the average log likelihood $L(\boldsymbol{\phi})$ will have high variance too. As a consequence, training can be unstable, and the trained model may not be an accurate posterior estimate. As we will see in the next chapter, SNPE-B typically produces less accurate results than SNPE-A, at least when SNPE-A doesn't terminate early.

In conclusion, although SNPE-B is an improvement over SNPE-A in terms of robustness and flexibility, I would argue that there is still scope for further developments in order to improve the performance and robustness of current methods. In the next chapter, I will discuss such a development, which I refer to as *Sequential Neural Likelihood*; SNL is an alternative to both SNPE-A and SNPE-B in that it targets the likelihood instead of the posterior. Nonetheless, SNPE remains an actively researched topic, so it would be reasonable to expect further improvements in the near future.

Chapter 4

Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows

The previous chapter discussed the challenge of likelihood-free inference, and described *Sequential Neural Posterior Estimation*, a method that estimates the unknown posterior with a neural density model. This chapter delves further into likelihood-free inference via neural density estimation, and introduces *Sequential Neural Likelihood*, an alternative approach that estimates the intractable likelihood instead. We begin with the paper *Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows*, which introduces SNL and is the main contribution of this chapter (section 4.1). We continue with a discussion of the paper, and a comparison with a related approach based on active learning (section 4.2). Finally, we conclude the part on likelihood-free inference with a summary and a discussion of this and the previous chapter (section 4.3).

4.1 The paper

This section presents the paper *Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows*, which is the main contribution of this chapter. The paper introduces *Sequential Neural Likelihood*, a new method for likelihood-free inference via neural density estimation that was designed to overcome some of the limitations of SNPE.

The paper was first published as a preprint on arXiv in May 2018. Afterwards, it was accepted for publication at the 22nd *International Conference on Artificial Intelligence and Statistics (AISTATS)* in April 2019. There were 1,111 submissions to AISTATS that year, of which 360 were accepted for publication.

Author contributions

The paper is co-authored by me, David Sterratt and Iain Murray. As the leading author, I developed the method, implemented the code, and performed the experiments. David Sterratt implemented the simulator for the Hodgkin–Huxley model, and advised on neuroscience matters. Iain Murray supervised the project and revised the final draft. The paper was written by me, except for sections A.4 and B.4 of the appendix which were written by David Sterratt.

Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows

George Papamakarios
University of Edinburgh

David C. Sterratt
University of Edinburgh

Iain Murray
University of Edinburgh

Abstract

We present Sequential Neural Likelihood (SNL), a new method for Bayesian inference in simulator models, where the likelihood is intractable but simulating data from the model is possible. SNL trains an autoregressive flow on simulated data in order to learn a model of the likelihood in the region of high posterior density. A sequential training procedure guides simulations and reduces simulation cost by orders of magnitude. We show that SNL is more robust, more accurate and requires less tuning than related neural-based methods, and we discuss diagnostics for assessing calibration, convergence and goodness-of-fit.

1 Introduction

In many areas of science and engineering, a natural way to model a stochastic system of interest is as a *simulator*, which may be controlled by potentially interpretable parameters and can be run forward to generate data. Such simulator models lend themselves naturally to modelling mechanistic processes such as particle collisions in high energy physics [1, 71], universe evolution in cosmology [2, 67], stochastic dynamical systems in ecology and evolutionary biology [64, 65, 84, 85], macroeconomic models in econometrics [3, 26], and biological neural networks in neuroscience [48, 62, 73]. Simulators are a powerful and flexible modelling tool; they can take the form of a computer graphics engine [45] or a physics engine [86], and they can become part of probabilistic programs [14, 16, 37].

Due to the wide applicability of simulator models, several tasks of interest can be framed as inference of the parameters of a simulator from observed data. For instance, inferring the parameters of physical theories given particle collisions can be the means for discovering new physics [11], and inferring the scene parameters of a graphics engine given a natural image can be the basis for computer vision [66, 87]. Hence, the development of accurate and efficient general-purpose inference methods for simulator models can have a profound impact in scientific discovery and artificial intelligence.

The fundamental difficulty in inferring the parameters of a simulator given data is the unavailability of the likelihood function. In Bayesian inference, posterior beliefs about parameters $\boldsymbol{\theta}$ given data $\mathbf{x}$ can be obtained by multiplying the likelihood $p(\mathbf{x} \mid \boldsymbol{\theta})$ with prior beliefs $p(\boldsymbol{\theta})$ and normalizing. However, calculating the likelihood $p(\mathbf{x} \mid \boldsymbol{\theta})$ of a simulator model for given parameters $\boldsymbol{\theta}$ and data $\mathbf{x}$ is computationally infeasible in general, and hence traditional likelihood-based Bayesian methods, such as variational inference [83] or Markov Chain Monte Carlo [54], are not directly applicable.

To overcome this difficulty, several methods for *likelihood-free inference* have been developed, such as Approximate Bayesian Computation [46] and Synthetic Likelihood [85], that require only the ability to generate data from the simulator. Such methods simulate the model repeatedly, and use the simulated data to build estimates of the parameter posterior. In general, the accuracy of likelihood-free inference improves as the number of simulations increases, but so does the computational cost, especially if the simulator is expensive to run. The challenge in developing new likelihood-free inference methods is to achieve a good trade-off between estimation accuracy and simulation cost.

In this paper we present *Sequential Neural Likelihood (SNL)*, a new likelihood-free method for Bayesian inference of simulator models, based on state-of-the-art conditional neural density estimation with autoregressive flows. The main idea of SNL is to train a Masked Autoregressive Flow [60] on simulated data in order to estimate the conditional probability density of data given parameters, which then serves as an accurate model of the likelihood function. During training, a Markov Chain Monte Carlo sampler selects the next

Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics (AISTATS) 2019, Naha, Okinawa, Japan. PMLR: Volume 89.

batch of simulations to run using the most up-to-date estimate of the likelihood function, leading to a reduction in the number of simulations of several orders of magnitude. To aid the practitioner in deploying SNL, we discuss practical issues such as implementation and tuning, diagnosing convergence, and assessing goodness-of-fit. Experimental results show that SNL is robust and well-calibrated, and that it outperforms existing state-of-the-art methods based on neural density estimation both in terms of posterior estimation accuracy and simulation cost.

2 Likelihood-free inference with neural density estimation

A simulator model is a computer program, which takes a vector of parameters $\boldsymbol{\theta}$, makes internal calls to a random number generator, and outputs a data vector $\mathbf{x}$. Implicitly, this procedure defines a conditional probability distribution $p(\mathbf{x} \,|\, \boldsymbol{\theta})$ which in general we cannot evaluate, but we can easily sample from by running the program. Given an observed data vector $\mathbf{x}_o$ and a prior distribution $p(\boldsymbol{\theta})$, we are interested in estimating the parameter posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x}_o) \propto p(\mathbf{x}_o \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})$.

This paper focuses on an approach to likelihood-free inference that is based on neural density estimation. A *conditional neural density estimator* is a parametric model $q_{\boldsymbol{\phi}}$ (such as a neural network) controlled by a set of parameters $\boldsymbol{\phi}$, which takes a pair of datapoints $(\mathbf{u}, \mathbf{v})$ and outputs a conditional probability density $q_{\boldsymbol{\phi}}(\mathbf{u} \,|\, \mathbf{v})$. Given a set of training data $\{\mathbf{u}_n, \mathbf{v}_n\}_{1:N}$ that are independent and identically distributed according to a joint probability density $p(\mathbf{u}, \mathbf{v})$, we can train $q_{\boldsymbol{\phi}}$ by maximizing the total log probability $\sum_n \log q_{\boldsymbol{\phi}}(\mathbf{u}_n \,|\, \mathbf{v}_n)$ with respect to $\boldsymbol{\phi}$. With enough training data, and with a sufficiently flexible model, $q_{\boldsymbol{\phi}}(\mathbf{u} \,|\, \mathbf{v})$ will learn to approximate the conditional $p(\mathbf{u} \,|\, \mathbf{v})$.

We can use a neural density estimator $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x})$ that models the conditional of parameters given data to approximate the posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ as follows. First, we obtain a set of samples $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}_{1:N}$ from the joint distribution $p(\boldsymbol{\theta}, \mathbf{x})$, by $\boldsymbol{\theta}_n \sim p(\boldsymbol{\theta})$ and $\mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta}_n)$ for $n = 1, \ldots, N$. Then, we train $q_{\boldsymbol{\phi}}$ using $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}_{1:N}$ as training data in order to obtain a global approximation of $p(\boldsymbol{\theta} \,|\, \mathbf{x})$. Finally, $p(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ can be simply estimated by $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$. In practice, a large number of simulations may be required for there to be enough training data in the vicinity of $\mathbf{x}_o$ in order to obtain an accurate posterior fit. However, running the simulator many times can be prohibitively expensive.

Sequential Neural Posterior Estimation is a strategy for reducing the number of simulations needed by conditional neural density estimation, originally proposed by Papamakarios and Murray [59] and further developed by Lueckmann et al. [43]. The name SNPE was first used for the method of Lueckmann et al. [43], but in this paper we will use it to refer to both methods, due to their close relationship. The main idea of SNPE is to generate parameter samples $\boldsymbol{\theta}_n$ from a proposal $\tilde{p}(\boldsymbol{\theta})$ instead of the prior $p(\boldsymbol{\theta})$ that makes generated data $\mathbf{x}_n$ more likely to be close to the observed datapoint $\mathbf{x}_o$. SNPE finds a good proposal $\tilde{p}(\boldsymbol{\theta})$ by training the estimator $q_{\boldsymbol{\phi}}$ over a number of rounds, whereby in each round $\tilde{p}(\boldsymbol{\theta})$ is taken to be the approximate posterior obtained in the round before. This sequential procedure converges rapidly and can be implemented with a relatively small number of simulations per round, which leads to massive savings in simulation cost. For its neural density estimator, SNPE uses a *Mixture Density Network* [7], which is a feedforward neural network that takes $\mathbf{x}$ as input and outputs the parameters of a Gaussian mixture over $\boldsymbol{\theta}$. To avoid overfitting due to the small number of training data per round, the MDN is trained with variational dropout [36].

The main issue with SNPE is that the proposal biases the approximation of the posterior. Since the parameter samples follow $\tilde{p}(\boldsymbol{\theta})$ instead of $p(\boldsymbol{\theta})$, the MDN will approximate $p(\mathbf{x}_o \,|\, \boldsymbol{\theta})\, \tilde{p}(\boldsymbol{\theta})$ instead of $p(\mathbf{x}_o \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})$ (up to normalization). Hence, an adjustment of either the learned posterior or the proposed samples must be made to account for sampling from the 'wrong' prior. In the variant of Papamakarios and Murray [59], which we refer to as SNPE-A, the learned posterior $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ is adjusted, by dividing it by $\tilde{p}(\boldsymbol{\theta})$ and multiplying it by $p(\boldsymbol{\theta})$. SNPE-A restricts $\tilde{p}(\boldsymbol{\theta})$ to be Gaussian; since $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ is a Gaussian mixture, the division by $\tilde{p}(\boldsymbol{\theta})$ can be done analytically. The problem with SNPE-A is that if $\tilde{p}(\boldsymbol{\theta})$ happens to have a smaller variance than any of the components of $q_{\boldsymbol{\phi}}(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$, the division yields a Gaussian with negative variance, from which the algorithm is unable to recover and thus is forced to terminate prematurely. In the variant of Lueckmann et al. [43], which we refer to as SNPE-B, the parameter samples $\boldsymbol{\theta}_n$ are adjusted, by assigning them weights $w_n = p(\boldsymbol{\theta}_n)/\tilde{p}(\boldsymbol{\theta}_n)$. During training, the weighted log likelihood $\sum_n w_n \log q_{\boldsymbol{\phi}}(\boldsymbol{\theta}_n \,|\, \mathbf{x}_n)$ is used instead of the total log likelihood. Compared to SNPE-A, this method does not require the proposal $\tilde{p}(\boldsymbol{\theta})$ to be Gaussian, and it does not suffer from negative variances. However, the weights can have high variance, which may result in high-variance gradients and instability during training.

3 Sequential Neural Likelihood

Our new method, *Sequential Neural Likelihood (SNL)*, avoids the bias introduced by the proposal, by opting to learn a model of the likelihood instead of the posterior. Let $\tilde{p}(\boldsymbol{\theta})$ be a proposal distribution over parameters

(not necessarily the prior) and let $p(\mathbf{x}\,|\,\boldsymbol{\theta})$ be the intractable likelihood of a simulator model. Consider a set of samples $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}_{1:N}$ obtained by $\boldsymbol{\theta}_n \sim \tilde{p}(\boldsymbol{\theta})$ and $\mathbf{x}_n \sim p(\mathbf{x}\,|\,\boldsymbol{\theta}_n)$, and define $\tilde{p}(\boldsymbol{\theta}, \mathbf{x}) = p(\mathbf{x}\,|\,\boldsymbol{\theta})\,\tilde{p}(\boldsymbol{\theta})$ to be the joint distribution of each pair $(\boldsymbol{\theta}_n, \mathbf{x}_n)$. Suppose we train a conditional neural density estimator $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$, which models the conditional of data given parameters, on the set $\{\boldsymbol{\theta}_n, \mathbf{x}_n\}_{1:N}$. For large N, maximizing the total log likelihood $\sum_n \log q_{\boldsymbol{\phi}}(\mathbf{x}_n\,|\,\boldsymbol{\theta}_n)$ is approximately equivalent to maximizing:

$$\mathbb{E}_{\tilde{p}(\boldsymbol{\theta},\mathbf{x})}(\log q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})) = \\ -\,\mathbb{E}_{\tilde{p}(\boldsymbol{\theta})}(D_{\mathrm{KL}}(p(\mathbf{x}\,|\,\boldsymbol{\theta})\,\|\,q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta}))) + \text{const}, \quad (1)$$

where $D_{\mathrm{KL}}(\cdot\,\|\,\cdot)$ is the Kullback–Leibler divergence. The above quantity attains its maximum when the KL is zero in the support of $\tilde{p}(\boldsymbol{\theta})$, i.e. when $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta}) = p(\mathbf{x}\,|\,\boldsymbol{\theta})$ for all $\boldsymbol{\theta}$ such that $\tilde{p}(\boldsymbol{\theta}) > 0$. Therefore, given enough simulations, a sufficiently flexible conditional neural density estimator will eventually approximate the likelihood in the support of the proposal, regardless of the shape of the proposal. In other words, as long as we do not exclude parts of the parameter space, the way we propose parameters does not bias learning the likelihood asymptotically. Unlike when learning the posterior, no adjustment is necessary to account for our proposing strategy.

In practice, for a moderate number of simulations, the proposal $\tilde{p}(\boldsymbol{\theta})$ controls where $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$ will be most accurate. In a parameter region where $\tilde{p}(\boldsymbol{\theta})$ is high, there will be a high concentration of training data, hence $p(\mathbf{x}\,|\,\boldsymbol{\theta})$ will be approximated better. Since we are ultimately interested in estimating the posterior $p(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$ for a specific datapoint $\mathbf{x}_o$, it makes sense to use a proposal that is high in regions of high posterior density, and low otherwise, to avoid expending simulations in regions that are not relevant to the inference task. Inspired by SNPE, we train $q_{\boldsymbol{\phi}}$ over multiple rounds, indexed by $r \geq 1$. Let $\hat{p}_{r-1}(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$ be the approximate posterior obtained in round $r-1$, and take $\hat{p}_0(\boldsymbol{\theta}\,|\,\mathbf{x}_o) = p(\boldsymbol{\theta})$. In round r, we generate a new batch of N parameters $\boldsymbol{\theta}_n$ from $\hat{p}_{r-1}(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$, and data $\mathbf{x}_n$ from $p(\mathbf{x}\,|\,\boldsymbol{\theta}_n)$. We then (re-)train $q_{\boldsymbol{\phi}}$ on all data generated in rounds 1 up to r, and set $\hat{p}_r(\boldsymbol{\theta}\,|\,\mathbf{x}_o) \propto q_{\boldsymbol{\phi}}(\mathbf{x}_o\,|\,\boldsymbol{\theta})\,p(\boldsymbol{\theta})$. This method, which we call *Sequential Neural Likelihood*, is detailed in Algorithm 1.

In round r, SNL effectively uses rN parameter samples from $\tilde{p}_r(\boldsymbol{\theta}) = \frac{1}{r}\sum_{i=0}^{r-1} \hat{p}_i(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$. As the amount of training data grows in each round, $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$ becomes a more accurate model of the likelihood in the region of high proposal density, and hence the approximate posterior $\hat{p}_r(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$ gets closer to the exact posterior. In turn, the proposal $\tilde{p}_r(\boldsymbol{\theta})$ also tends to the exact posterior, and therefore most of the simulations in later rounds come from parameter regions of high posterior density. In Section 5 we see that focusing on high posterior parameters massively reduces the number of simulations. Finally, unlike SNPE which trains only on simulations from the latest round, SNL trains on all simulations obtained up to each round.

Algorithm 1: Sequential Neural Likelihood (SNL)

Input : observed data $\mathbf{x}_o$, estimator $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$, number of rounds R, simulations per round N
Output : approximate posterior $\hat{p}(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$

set $\hat{p}_0(\boldsymbol{\theta}\,|\,\mathbf{x}_o) = p(\boldsymbol{\theta})$ and $\mathcal{D} = \{\}$
for $r = 1 : R$ **do**
  for $n = 1 : N$ **do**
    sample $\boldsymbol{\theta}_n \sim \hat{p}_{r-1}(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$ with MCMC
    simulate $\mathbf{x}_n \sim p(\mathbf{x}\,|\,\boldsymbol{\theta}_n)$
    add $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ into $\mathcal{D}$
  (re-)train $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$ on $\mathcal{D}$ and set $\hat{p}_r(\boldsymbol{\theta}\,|\,\mathbf{x}_o) \propto q_{\boldsymbol{\phi}}(\mathbf{x}_o\,|\,\boldsymbol{\theta})\,p(\boldsymbol{\theta})$
return $\hat{p}_R(\boldsymbol{\theta}\,|\,\mathbf{x}_o)$

In general, we are free to choose any neural density estimator $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$ that is suitable for the task at hand. Because we are interested in a general-purpose solution, we propose taking $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$ to be a conditional *Masked Autoregressive Flow* [60], which has been shown to perform well in a variety of general-purpose density estimation tasks. MAF represents $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta})$ as a transformation of a standard Gaussian density $\mathcal{N}(\mathbf{0}, \mathbf{I})$ through a series of K autoregressive functions $f_1, \ldots, f_K$ each of which depends on $\boldsymbol{\theta}$, that is:

$$\mathbf{x} = \mathbf{z}_{\mathbf{K}} \quad \text{where} \quad \begin{array}{l} \mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\ \mathbf{z}_k = f_k(\mathbf{z}_{k-1}, \boldsymbol{\theta}). \end{array} \quad (2)$$

Each f_k is a bijection with a lower-triangular Jacobian matrix, and is implemented by a *Masked Autoencoder for Distribution Estimation* [23] conditioned on $\boldsymbol{\theta}$. By change of variables, the conditional density is given by $q_{\boldsymbol{\phi}}(\mathbf{x}\,|\,\boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_0\,|\,\mathbf{0}, \mathbf{I}) \prod_k \left|\det\left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}}\right)\right|^{-1}$.

In order to sample from the approximate posterior $\hat{p}(\boldsymbol{\theta}\,|\,\mathbf{x}_o) \propto q_{\boldsymbol{\phi}}(\mathbf{x}_o\,|\,\boldsymbol{\theta})\,p(\boldsymbol{\theta})$, we use Markov Chain Monte Carlo (MCMC) in the form of Slice Sampling with axis-aligned updates [55]. The Markov chain is initialized with a sample from the prior and it persists across rounds; that is, in every round other than the first, the initial state of the chain is the same as its last state in the round before. At the beginning of each round, the chain is burned-in for 200 iterations, in order to adapt to the new approximate posterior. This scheme requires no tuning, and we found that it performed robustly across our experiments, so we recommend it as a default for general use, at least for tens of parameters that do not have pathologically strong correlations.

For parameter spaces of larger dimensionality, other MCMC schemes could be considered, such as Hamiltonian Monte Carlo [56].

4 Related work

Approximate Bayesian Computation [4, 5, 8, 42, 46]. ABC is a family of mainly non-parametric methods that repeatedly simulate the model, and reject simulations that do not reproduce the observed data. In practice, to reduce rejection rates to manageable levels, (a) lower-dimensional features (or summary statistics) are used instead of raw data, and (b) simulations are accepted whenever simulated features are within a distance ϵ from the observed features. A basic implementation of ABC would simulate parameters from the prior; more sophisticated variants such as *Markov Chain Monte Carlo ABC* [47, 51, 69] and *Sequential Monte Carlo ABC* [6, 10, 70, 76] guide future simulations based on previously accepted parameters. An issue with ABC in general is that in practice the required number of simulations increases dramatically as ϵ becomes small, which, as we shall see in Section 5, can lead to an unfavourable trade-off between estimation accuracy and simulation cost. Advanced ABC algorithms that work for $\epsilon = 0$ exist [27], but require the simulator to be differentiable.

Learning the posterior. Another approach to likelihood-free inference is learning a parametric model of the posterior from simulations. *Regression Adjustment* [5, 9] employs a parametric regressor (such as a linear model or a neural network) to learn the dependence of parameters $\boldsymbol{\theta}$ given data $\mathbf{x}$ in order to correct posterior parameter samples (obtained by e.g. ABC). *Gaussian Copula ABC* [40] estimates the posterior with a parametric Gaussian copula model. *Variational Likelihood-Free Inference* [52, 77, 78] trains a parametric posterior model by maximizing the (here intractable) variational lower bound; either the bound or its gradients are stochastically estimated from simulations. Parametrically learning the posterior is the target of *Sequential Neural Posterior Estimation* [43, 59], which SNL directly builds on, and which was discussed in detail in Section 2. Beyond SNPE, conditional neural density estimators have been used to learn the posterior from simulations in graphical models [53, 58], and universal probabilistic programs [38].

Learning the likelihood. Similarly to SNL, a body of work has focused on parametrically approximating the intractable likelihood function. *Synthetic Likelihood* [20, 57, 63, 85] estimates the mean $\mathbf{m}_{\boldsymbol{\theta}}$ and covariance matrix $\mathbf{S}_{\boldsymbol{\theta}}$ of a batch of data $\mathbf{x}_n \sim p(\mathbf{x} \,|\, \boldsymbol{\theta})$ sampled at a given $\boldsymbol{\theta}$, and then approximates $p(\mathbf{x}_o \,|\, \boldsymbol{\theta}) \approx \mathcal{N}(\mathbf{x}_o \,|\, \mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}})$. Non-Gaussian likelihood approximations are also possible, e.g. saddlepoint approximations [22]. Typically, SL would be run as an inner loop in the context of an MCMC sampler. *Gaussian Process Surrogate ABC* [49] employs a Gaussian process to model the dependence of $(\mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}})$ on $\boldsymbol{\theta}$, and uses the uncertainty in the GP to decide whether to run more simulations to estimate $(\mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}})$. Other predecessors to SNL include approximating the likelihood as a linear-Gaussian model [39], or as a mixture of Gaussian copulas with marginals modelled by mixtures of experts [21].

Learning the likelihood ratio. Rather than learning the likelihood $p(\mathbf{x} \,|\, \boldsymbol{\theta})$, an alternative approach is learning the likelihood ratio $p(\mathbf{x} \,|\, \boldsymbol{\theta}_1)/p(\mathbf{x} \,|\, \boldsymbol{\theta}_2)$ [11, 12, 15, 30, 61, 74], or the likelihood-to-marginal ratio $p(\mathbf{x} \,|\, \boldsymbol{\theta})/p(\mathbf{x})$ [19, 33, 77]. In practice, this can be done by training a classifier to discriminate between data simulated at $\boldsymbol{\theta}_1$ vs $\boldsymbol{\theta}_2$ for the likelihood ratio, or between data simulated at $\boldsymbol{\theta}$ vs random parameter samples from the prior for the likelihood-to-marginal ratio. The likelihood ratio can be used directly for Bayesian inference (e.g. by multiplying it with the prior and then sampling with MCMC), for hypothesis testing [11, 15], or for estimating the variational lower bound in the context of variational inference [77]. We note that the strategy for guiding simulations proposed in this paper can (at least in principle) be used with likelihood-ratio estimation too; it might be possible that large computational savings can be achieved as in SNL.

Guiding simulations. Various approaches have been proposed for guiding simulations in order to reduce simulation cost, typically by making use of the observed data $\mathbf{x}_o$ in some way. *Optimization Monte Carlo* [50] directly optimizes parameter samples so as to generate data similar to $\mathbf{x}_o$. *Bayesian Optimization Likelihood-free Inference* [29] uses Bayesian optimization to find parameter samples that minimize the distance $\|\mathbf{x} - \mathbf{x}_o\|$. An active-learning-style approach is to use the Bayesian uncertainty in the posterior estimate to choose what simulation to run next [34, 44]; in this scheme, the next simulation is chosen to maximally reduce the Bayesian uncertainty. Similarly to *Sequential Neural Posterior Estimation* [43, 59], SNL selects future simulations by proposing parameters from preliminary approximations to the posterior, which in Section 5 is shown to reduce simulation cost significantly.

5 Experiments

5.1 Setup

In all experiments, we used a Masked Autoregressive Flow (MAF) with 5 autoregressive layers, each of which has two hidden layers of 50 units each and tanh nonlinearities. Our design and training choices follow

closely the reference implementation [60]. We used batch normalization [32] between autoregressive layers, and trained MAF by stochastically maximizing the total log likelihood using Adam [35] with a minibatch size of 100 and a learning rate of 10^{-4}. We used 1000 simulations in each round of SNL, 5% of which were randomly selected to be used as a validation set; we stopped training if validation log likelihood did not improve after 20 epochs. No other form of regularization was used other than early stopping. These settings were held constant and performed robustly across all experiments, which is evidence for the robustness of SNL to parameter tuning. We recommend these settings as defaults for general use.

We compare SNL to the following algorithms:

Neural Likelihood (NL). By this we refer to training a MAF (of the same architecture as above) on N simulations from the prior. This is essentially SNL without simulation guiding; we use it as a control to assess the benefit of SNL's guiding strategy. We vary N from 10^3 to 10^6 (or more if the simulation cost permits it), and plot the performance for each N.

SNPE-A [59]. We use 1000 simulations in each of the proposal-estimating rounds, and 2000 simulations in the posterior-estimating round (the final round). In all experiments, SNPE-A uses a Mixture Density Network [7] with two hidden layers of 50 units each, 8 mixture components, and tanh nonlinearities. We chose the MDN to have roughly as many parameters as MAF in SNL. The MDN is trained for 1000 epochs per round, except for the final round which uses 5000 epochs, and is regularized with variational dropout [36].

SNPE-B [43]. We use 1000 simulations in each round. The same MDN as in SNPE-A is used, and it is trained for 1000 epochs per round using variational dropout.

Synthetic Likelihood (SL) [85]. Our implementation uses axis-aligned Slice Sampling [55], where the intractable likelihood is approximated by a Gaussian fitted to a batch of N simulations run on the fly at each visited parameter $\boldsymbol{\theta}$. We vary N from 10 to 100–1000 (depending on what the simulation cost of each experiment permits) and plot the performance of SL for each N. We run Slice Sampling until we obtain 1000 posterior samples.

Sequential Monte Carlo ABC (SMC-ABC). We use the version of Beaumont et al. [6]. We use 1000 particles, and resample the population if the effective sample size falls below 50%. We set the initial ϵ such that the acceptance probability in the first round is at least 20%, and decay ϵ by a factor of 0.9 per round.

We chose SNPE-A/B because they are the most related methods, and because they achieve state-of-the-art results in the literature; the comparison with them is intended to establish the competitiveness of SNL. The versions of SL and SMC-ABC we use here are not state-of-the-art, but they are robust and widely-used, and are intended as baselines. The comparison with them is meant to demonstrate the gap between state-of-the-art neural-based methods and off-the-shelf, commonly-used alternatives.

5.2 Results

We demonstrate SNL in four cases: (a) a toy model with complex posterior, (b) an M/G/1 queue model, (c) a Lotka–Volterra model from ecology, and (d) a Hodgkin–Huxley model of neural activity from neuroscience. The first two models are fast to simulate, and are intended as toy demonstrations. The last two models are relatively slow to simulate (as they involve numerically solving differential equations), and are illustrative of real-world problems of interest. In the last two models, the computational cost of training the neural networks for SNPE-A/B and SNL is negligible compared to the cost of simulating training data.

A detailed description of the models and the full set of results are in Appendices A and B. Code that reproduces the experiments with detailed user instructions can be found at `https://github.com/gpapamak/snl`.

A toy model with complex posterior. We consider the following model, where $\boldsymbol{\theta}$ is 5-dimensional, and $\mathbf{x}$ is a set of four 2-dimensional points (or an 8-dimensional vector) sampled from a Gaussian whose mean $\mathbf{m}_{\boldsymbol{\theta}}$ and covariance matrix $\mathbf{S}_{\boldsymbol{\theta}}$ are functions of $\boldsymbol{\theta}$:

$$\theta_i \sim \mathcal{U}(-3, 3) \quad \text{for} \quad i = 1, \dots, 5 \tag{3}$$

$$\mathbf{m}_{\boldsymbol{\theta}} = (\theta_1, \theta_2) \tag{4}$$

$$s_1 = \theta_3^2, \quad s_2 = \theta_4^2, \quad \rho = \tanh(\theta_5) \tag{5}$$

$$\mathbf{S}_{\boldsymbol{\theta}} = \begin{pmatrix} s_1^2 & \rho s_1 s_2 \\ \rho s_1 s_2 & s_2^2 \end{pmatrix} \tag{6}$$

$$\mathbf{x} = (\mathbf{x}_1, \dots, \mathbf{x}_4) \quad \text{where} \quad \mathbf{x}_j \sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}}). \tag{7}$$

The likelihood is $p(\mathbf{x} \,|\, \boldsymbol{\theta}) = \prod_j \mathcal{N}(\mathbf{x}_j \,|\, \mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}})$. Despite the model's simplicity, the posterior is complex and non-trivial: it has four symmetric modes (due to squaring), and vertical cut-offs (due to the uniform prior). This example illustrates that the posterior can be complicated even if the prior and the likelihood are composed entirely of simple operations (which is a common situation e.g. in probabilistic programming). In such situations, approximating the likelihood can be simpler than approximating the posterior.

Figure 1a shows the Maximum Mean Discrepancy (MMD) [28] between the approximate posterior of each method and the true posterior vs the total number of simulations used. SNL achieves the best trade-off

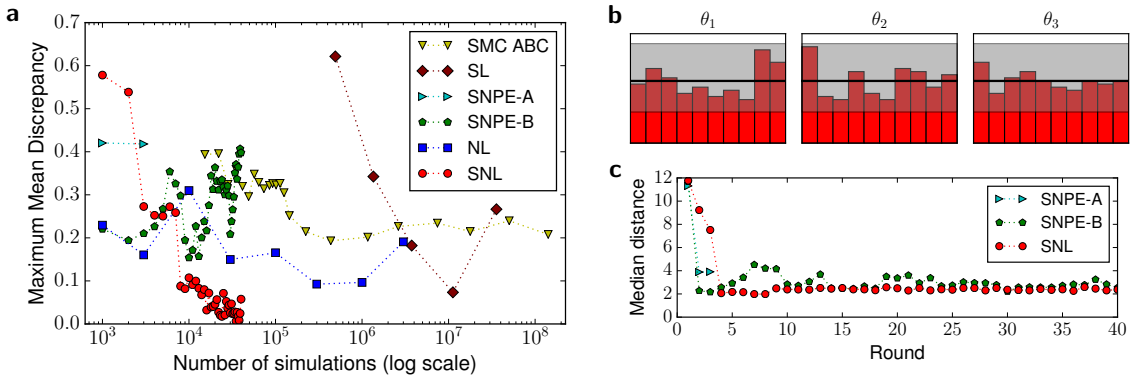


Figure 1: A toy model with complex posterior. **a**: Accuracy vs simulation cost: bottom left is best. **b**: Calibration test for SNL, histogram outside gray band indicates poor calibration. **c**: Median distance from simulated to observed data.

between accuracy and simulation cost, and achieves the most accurate approximation to the posterior overall. SNPE-A fails in the second round due to the variance of the proposal becoming negative (hence there are only two points in the graph) and SNPE-B experiences high variability; SNL is significantly more robust in comparison. SMC-ABC and SL require orders of magnitude more simulations than the sequential neural methods.

To assess whether SNL is well-calibrated, we performed a simulation-based calibration test [75]: we generated 200 pairs $(\boldsymbol{\theta}_n, \mathbf{x}_n)$ from the joint $p(\boldsymbol{\theta}, \mathbf{x})$, used SNL to approximate the posterior for each $\mathbf{x}_n$, obtained 9 close-to-independent samples from each posterior, and calculated the rank statistic of each parameter θ_{ni} for $i = 1, \ldots, 5$ in the corresponding set of posterior samples. If SNL is well-calibrated, the distribution of each rank statistic must be uniform. The histograms of the first three rank statistics are shown in Figure 1b, with a gray band showing the expected variability of a uniform histogram. The test does not find evidence of any gross mis-calibration, and suggests that SNL performs consistently across multiple runs (here 200).

Another diagnostic is shown in Figure 1c, where we plot the median distance between simulated and observed data for each round. From this plot we can assess convergence, and determine the minimum number of rounds to run for. SNL has lower median distance compared to SNPE-B, which is evidence that SNPE-B has not estimated the posterior accurately enough (as also shown in the left plot).

M/G/1 queue model [68]. The model describes a server processing customers waiting in a queue, and has parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$. The time it takes to serve a customer is uniformly distributed in $[\theta_1, \theta_2]$, and the time between customer arrivals is exponentially distributed with rate θ_3. The data $\mathbf{x}$ is 5 equally spaced quantiles of the distribution of inter-departure times. The model can be easily simulated, but the likelihood $p(\mathbf{x} \mid \boldsymbol{\theta})$ is intractable. Our experimental setup follows Papamakarios and Murray [59].

The trade-off between accuracy and simulation cost is shown in Figure 2a. Here we do not have access to the true posterior; instead we plot the negative log probability of the true parameters, obtained by kernel density estimation on posterior samples, vs number of simulations. SNL recovers the true parameters faster and more accurately than all other methods. Although high posterior probability of true parameters is not enough evidence that the posterior is correctly calibrated (e.g. it could be that the approximate posterior happens to be centred at the right value, but is overconfident), the calibration test (Figure 2b) suggests that SNL is indeed well-calibrated.

Another diagnostic possible with SNL is to check how well $q_{\boldsymbol{\phi}}(\mathbf{x} \mid \boldsymbol{\theta})$ fits the data distribution $p(\mathbf{x} \mid \boldsymbol{\theta})$ for a certain value of $\boldsymbol{\theta}$. To do this we (a) generate N independent samples from $p(\mathbf{x} \mid \boldsymbol{\theta})$ by running the simulator, (b) generate N independent samples from $q_{\boldsymbol{\phi}}(\mathbf{x} \mid \boldsymbol{\theta})$ using Equation (2), and (c) calculate the Maximum Mean Discrepancy [28] between the two sets of samples. This is shown in Figure 2c using the true parameters, where we compare with NL and a baseline Gaussian directly fitted to the samples from $p(\mathbf{x} \mid \boldsymbol{\theta})$. This plot can be used to assess the performance of SNL, and also determine how many rounds to run SNL for. We note that this kind of diagnostic is not possible with methods that approximate the posterior or the likelihood ratio instead of the likelihood.

Lotka–Volterra population model [84]. This is a Markov jump process that models the interaction of a population of predators with a population of prey. It is a classic model of oscillating populations. It has four

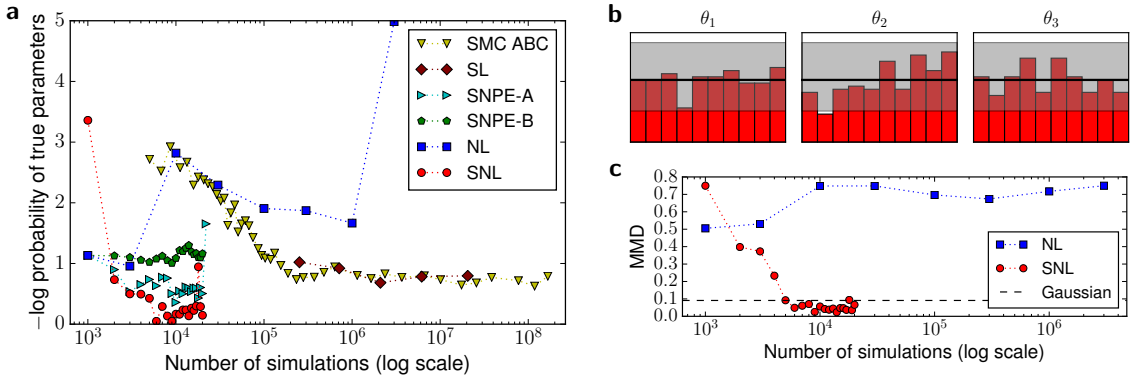


Figure 2: M/G/1 queue model. **a**: Accuracy vs simulation cost: bottom left is best. **b**: Calibration test for SNL, histogram outside gray band indicates poor calibration. **c**: Likelihood goodness-of-fit vs simulation cost, calculated at true parameters.

parameters $\boldsymbol{\theta}$, which control the rate of (a) predator births, (b) predator deaths, (c) prey births, and (d) predator-prey interactions. The process can be simulated exactly (by using e.g. the Gillespie algorithm [24]), but its likelihood is intractable. In our experiments we follow the setup of Papamakarios and Murray [59], where $\mathbf{x}$ is 9 features of the population timeseries.

Figure 3a shows negative log probability of true parameters vs simulation cost. SNL and SNPE-A perform the best, whereas SNPE-B is less accurate. Running the calibration test for SNL, using parameters drawn from a broad prior, shows the procedure is sometimes over-confident (Figure 3b, left plot). In this test, many of the 'true' parameters considered corresponded to uninteresting models, where both populations died out quickly, or the prey population diverged. In an application we want to know if the procedure is well-behaved for the sorts of parameters we seem to have. We ran a calibration test for an alternative prior, constrained to parameters that give the oscillating behaviour observed in interesting data. This test suggests that the calibration is reasonable when modelling oscillating data (Figure 3b, right plot). If we had still observed calibration problems we would have investigated using larger neural networks and/or longer MCMC runs in SNL. Figure 3c shows the median distance between the simulated data and the observed data for each round of SNPE-A, SNPE-B and SNL. We see that SNPE-A and SNL have a lower median distance compared to SNPE-B, which suggests that SNPE-B has not estimated the posterior accurately enough.

Hodgkin–Huxley neuron model [31]. This model describes how the electrical potential measured across the cell membrane of a neuron varies over time as a function of current injected through an electrode. We used a model of a regular-spiking cortical pyramidal cell [62]; the model is described by a set of five coupled ordinary differential equations, which we solved numerically using NEURON [13]. This is a challenging problem of practical interest in neuroscience [17, 43], whose task is to infer 12 parameters describing the function of the neuron from features of the membrane potential timeseries. We followed the setup of Lueckmann et al. [43], and we used 18 features extracted from the timeseries as data $\mathbf{x}$.

Figure 4a shows negative log probability of true parameters vs simulation cost; SNL outperforms all other methods. SNPE-A fails in the second round due to the variance of the proposal becoming negative. The calibration test in Figure 4b suggests that SNL is well-calibrated. Figure 4c shows goodness-of-fit between MAF and the true likelihood as measured by MMD; SNL converges faster than NL to the true likelihood. The comparison with a baseline Gaussian fit is evidence that SNL has not fully converged, which indicates that running SNL for longer may improve results further.

6 Discussion

Performance and robustness of SNL. Our experiments show that SNL is accurate, efficient and robust. The comparison between SNL and NL demonstrates that guiding the simulations is crucial in achieving a good likelihood fit in the region of interest with a reasonable number of simulations. The comparison between SNL and SMC-ABC shows that flexible parametric models can be more accurate and cost-effective compared to non-parametric alternatives. The comparison with SNPE shows that SNL is more robust; SNPE-A failed in two out of four cases, and SNPE-B exhibited high variability. We used the same MAF architecture and training hyperparameters in all our experiments, which shows that a flexible neural architecture can be broadly applicable without extensive

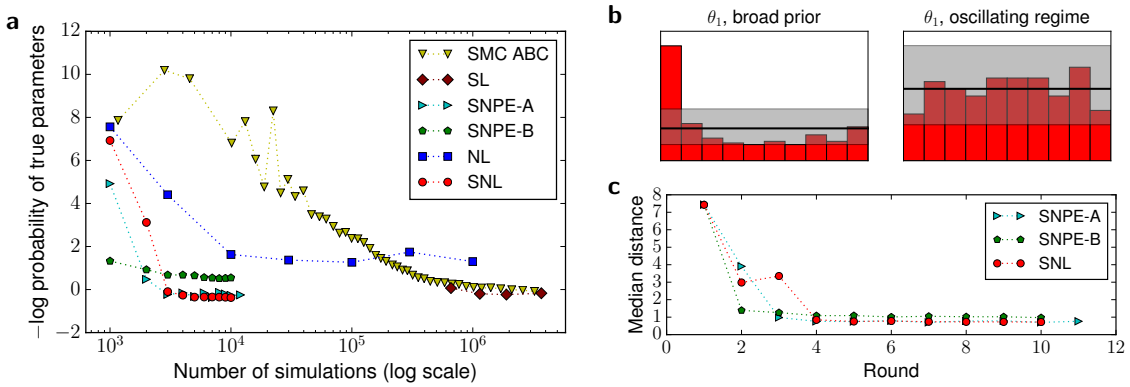


Figure 3: Lotka–Volterra model. **a**: Accuracy vs simulation cost: bottom left is best. **b**: Calibration tests for SNL, histogram outside gray band indicates poor calibration. Calibration depends on data regime (see main text). **c**: Median distance from simulated to observed data.

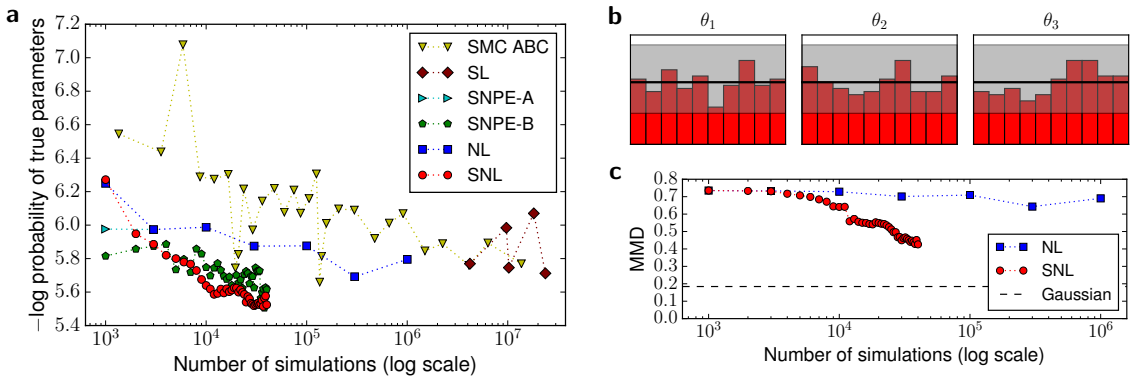


Figure 4: Hodgkin–Huxley model. **a**: Accuracy vs simulation cost: bottom left is best. **b**: Calibration test for SNL, histogram outside gray band indicates poor calibration. **c**: Likelihood goodness-of-fit vs simulation cost, calculated at true parameters.

problem-specific tuning. We also discussed a number of diagnostics that can be used with SNL to determine the number of rounds to run it for, assess calibration, diagnose convergence and check goodness-of-fit.

Scaling to high-dimensional data. Likelihood-free inference becomes challenging when the dimensionality of the data $\mathbf{x}$ is large, which in practice necessitates the use of low-dimensional features (or summary statistics) in place of the raw data. SNL relies on estimating the density of the data, which is a hard problem in high dimensions. A potential strategy for scaling SNL up to high dimensions is exploiting the structure of the data. For instance, if $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$ is a dataset of a large number of i.i.d. datapoints, we could decompose the likelihood as $p(\mathbf{x} \,|\, \boldsymbol{\theta}) = \prod_n p(\mathbf{x}_n \,|\, \boldsymbol{\theta})$ and only learn a model $q_{\boldsymbol{\phi}}(\mathbf{x}_n \,|\, \boldsymbol{\theta})$ of the density in the lower-dimensional space of individual points. By exploiting data structure, neural density estimators have been shown to accurately model high-dimensional data such as images [18, 80, 81] and raw audio [79, 82]; SNL could easily incorporate such or further advances in neural density estimation for high-dimensional structured objects.

Learning the likelihood vs the posterior. A general question is whether it is preferable to learn the posterior or the likelihood; SNPE learns the posterior, whereas SNL targets the likelihood. As we saw, learning the likelihood can often be easier than learning the posterior, and it does not depend on the choice of proposal, which makes learning easier and more robust. Moreover, a model of the likelihood can be reused with different priors, and is in itself an object of interest that can be used for identifiability analysis [41] or hypothesis testing [11, 15]. On the other hand, methods such as SNPE return a parametric model of the posterior directly, whereas a further inference step (e.g. variational inference or MCMC) is needed on top of SNL to obtain a posterior estimate, which introduces further computational cost and approximation error. Ultimately, the best approach depends on the problem and application at hand.

Acknowledgments

We thank Kyle Cranmer and Michael Gutmann for useful discussions, and James Ritchie for pointing us to simulation-based calibration [75]. George Papamakarios was supported by the Centre for Doctoral Training in Data Science, funded by EPSRC (grant EP/L016427/1) and the University of Edinburgh, and by Microsoft Research through its PhD Scholarship Programme. David C. Sterratt received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 785907.

A Detailed description of simulator models used in experiments

A.1 A toy problem with complex posterior

This toy model illustrates that even simple models can have non-trivial posteriors. The model has 5 parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_5)$ sampled from a uniform prior as follows:

$$\theta_i \sim \mathcal{U}(-3,\, 3) \quad \text{for} \quad i = 1, \ldots, 5. \tag{8}$$

Given parameters $\boldsymbol{\theta}$, the data $\mathbf{x}$ is generated as follows:

$$\mathbf{m}_{\boldsymbol{\theta}} = (\theta_1, \theta_2) \tag{9}$$
$$s_1 = \theta_3^2 \tag{10}$$
$$s_2 = \theta_4^2 \tag{11}$$
$$\rho = \tanh(\theta_5) \tag{12}$$
$$\mathbf{S}_{\boldsymbol{\theta}} = \begin{pmatrix} s_1^2 & \rho s_1 s_2 \\ \rho s_1 s_2 & s_2^2 \end{pmatrix} \tag{13}$$
$$\mathbf{x}_j \sim \mathcal{N}(\mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}}) \quad \text{for} \quad j = 1, \ldots, 4 \tag{14}$$
$$\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_4). \tag{15}$$

The data $\mathbf{x}$ is 8-dimensional. The likelihood is:

$$p(\mathbf{x} \,|\, \boldsymbol{\theta}) = \prod_{j=1}^{4} \mathcal{N}(\mathbf{x}_j \,|\, \mathbf{m}_{\boldsymbol{\theta}}, \mathbf{S}_{\boldsymbol{\theta}}). \tag{16}$$

In our experiments, we took the ground truth parameters to be:

$$\boldsymbol{\theta}^* = (0.7,\ \ -2.9,\ \ -1,\ \ -0.9,\ \ 0.6), \tag{17}$$

and simulated the model with parameters $\boldsymbol{\theta}^*$ to get observed data $\mathbf{x}_o$.

A.2 M/G/1 queue model

The M/G/1 queue model [68] describes how a server processes a queue of arriving customers. Our experimental setup follows Papamakarios and Murray [59]. There are 3 parameters $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)$ sampled from a uniform prior as follows:

$$\theta_1 \sim \mathcal{U}(0, 10) \tag{18}$$
$$\theta_2 - \theta_1 \sim \mathcal{U}(0, 10) \tag{19}$$
$$\theta_3 \sim \mathcal{U}(0, 1/3). \tag{20}$$

Let I be the total number of customers, s_i be the time the server takes to serve customer i, a_i be the time customer i arrived, and d_i be the time customer i departed. Take $a_0 = d_0 = 0$. The M/G/1 queue model is described by:

$$s_i \sim \mathcal{U}(\theta_1, \theta_2) \tag{21}$$
$$a_i - a_{i-1} \sim \text{Exp}(\theta_3) \tag{22}$$
$$d_i - d_{i-1} = s_i + \max(0, a_i - d_{i-1}). \tag{23}$$

In our experiments we used $I = 50$. The data $\mathbf{x}$ is 5-dimensional, and is obtained by (a) calculating the 0th, 25th, 50th, 75th and 100th quantiles of the set of inter-departure times $\{d_i - d_{i-1}\}_{1:I}$, and (b) linearly transforming the quantiles to have approximately zero mean and unit covariance matrix. The parameters of the linear transformation were determined by a pilot run. We took the ground truth parameters to be:

$$\boldsymbol{\theta}^* = (1,\ \ 5,\ \ 0.2), \tag{24}$$

and simulated the model with parameters $\boldsymbol{\theta}^*$ to get observed data $\mathbf{x}_o$.

A.3 Lotka–Volterra population model

The Lotka–Volterra model [84] is a Markov jump process describing the evolution of a population of predators interacting with a population of prey, and has four parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_4)$. Let X be the number of predators, and Y be the number of prey. According to the model, the following can take place:

- With rate $\exp(\theta_1)XY$ a predator may be born, increasing X by one.
- With rate $\exp(\theta_2)X$ a predator may die, decreasing X by one.
- With rate $\exp(\theta_3)Y$ a prey may be born, increasing Y by one.
- With rate $\exp(\theta_4)XY$ a prey may be eaten by a predator, decreasing Y by one.

Our experimental setup follows that of Papamakarios and Murray [59]. We used initial populations $X = 50$ and $Y = 100$. We simulated the model using the Gillespie algorithm [24] for a total of 30 time units. We recorded the two populations every 0.2 time units,

which gives two timeseries of 151 values each. The data $\mathbf{x}$ is 9-dimensional, and corresponds to the following timeseries features:

- The mean of each timeseries.
- The log variance of each timeseries.
- The autocorrelation coefficient of each timeseries at lags 0.2 and 0.4 time units.
- The cross-correlation coefficient between the two timeseries.

Each feature was normalized to have approximately zero mean and unit variance based on a pilot run. The ground truth parameters were taken to be:

$$\boldsymbol{\theta}^* = (\log 0.01,\ \ \log 0.5,\ \ \log 1,\ \ \log 0.01), \tag{25}$$

and the observed data $\mathbf{x}_o$ were generated from a simulation of the model at $\boldsymbol{\theta}^*$. In our experiments we used two priors: (a) a broad prior defined by:

$$p_{\text{broad}}(\boldsymbol{\theta}) \propto \prod_{i=1}^{4} I(-5 \leq \theta_i \leq 2), \tag{26}$$

and (b) a prior corresponding to the oscillating regime, defined by:

$$p_{\text{osc}}(\boldsymbol{\theta}) \propto \mathcal{N}\big(\boldsymbol{\theta} \,|\, \boldsymbol{\theta}^*, 0.5^2\big)\, p_{\text{broad}}(\boldsymbol{\theta}). \tag{27}$$

A.4 Hodgkin–Huxley cortical pyramidal neuron model

In neuroscience, the formalism developed by Hodgkin–Huxley in their classic model of the squid giant axon [31] is used to model many different types of neuron. In our experiments, we used a slightly modified version of a regular-spiking cortical pyramidal cell [62], for which NEURON [13] simulation code is available in ModelDB.[1] The model is formulated as a set of five coupled ordinary differential equations (ODEs) and describes how the electrical potential $V(t)$ measured across the neuronal cell membrane varies over time as a function of current $I_{\text{e}}(t)$ injected through an electrode. In essence, the membrane is a capacitor punctuated by conductances formed by multiple types of ion channel through which currents flow. The currents charge and discharge the membrane capacitance, causing the membrane potential to change, as described by the first ODE, the membrane equation:

$$C_{\text{m}} \frac{\mathrm{d}V}{\mathrm{d}t} = -I_\ell - I_{\text{Na}} - I_{\text{K}} - I_{\text{M}} - I_{\text{e}}. \tag{28}$$

[1] `https://senselab.med.yale.edu/ModelDB/ShowModel.cshtml?model=123623`

Here, $C_{\text{m}} = 1\,\mu\text{F}\,\text{cm}^{-2}$ is the specific membrane capacitance, and I_ℓ, I_{Na}, I_{K} and I_{M} are the ionic currents flowing through 'leak' channels, sodium channels, potassium delayed-rectifier channels and M-type potassium channels respectively. Each ionic current depends on a conductance that corresponds to how many channels are open, and on the difference between the membrane potential and an equilibrium potential. For example, for the leak current:

$$I_\ell = g_\ell\,(V - E_\ell), \tag{29}$$

where g_ℓ is the leak conductance, and E_ℓ is the leak equilibrium potential. Here the conductance g_ℓ is constant through time, but for the sodium, potassium and M-type channels, the conductances vary over time, as described by the product of a fixed conductance and time-varying state variables:

$$I_{\text{Na}} = \overline{g}_{\text{Na}}\, m^3 h\,(V - E_{\text{Na}}) \tag{30}$$

$$I_{\text{K}} = \overline{g}_{\text{K}}\, n^4\,(V - E_{\text{K}}) \tag{31}$$

$$I_{\text{M}} = \overline{g}_{\text{M}}\, p\,(V - E_{\text{K}}). \tag{32}$$

Here, $\overline{g}_{\text{Na}}$, $\overline{g}_{\text{K}}$ and $\overline{g}_{\text{M}}$ are the per-channel maximum conductances, m, h, n and p are state variables that range between 0 and 1, and E_{Na} and E_{K} are the sodium and potassium reversal potentials. The state variables evolve according to differential equations of the form first introduced by Hodgkin and Huxley [31]:

$$\frac{\mathrm{d}x}{\mathrm{d}t} = \alpha_x(V)(1 - x) - \beta_x(V)\, x \quad \text{for } x \in \{m, h, n\} \tag{33}$$

$$\frac{\mathrm{d}p}{\mathrm{d}t} = \frac{p - p_\infty(V)}{\tau_{\text{p}}(V)}, \tag{34}$$

where $\alpha_x(V)$, $\beta_x(V)$, $p_\infty(V)$ and $\tau_{\text{p}}(V)$ are nonlinear functions of the membrane potential. We use the published equations [62] for $\alpha_{\text{m}}(V)$, $\beta_{\text{m}}(V)$, $\alpha_{\text{h}}(V)$, $\beta_{\text{h}}(V)$ and $\alpha_{\text{n}}(V)$, which contain a parameter V_{T}, and the published equations for $p_\infty(V)$ and $\tau_{\text{p}}(V)$, the latter of which contains a parameter $\tau_{\max}$. We use a generalized version of $\beta_{\text{n}}(V)$:

$$\beta_{\text{n}}(V) = k_{\beta\text{n}1} \exp\!\left(-\frac{V - V_{\text{T}} - 10}{k_{\beta\text{n}2}}\right) \tag{35}$$

in which $k_{\beta\text{n}1}$ and $k_{\beta\text{n}2}$ are adjustable parameters (rather than $0.5\,\text{ms}^{-1}$ and $40\,\text{mV}$). In order to simulate the model, we use NEURON [13] to solve the ODEs numerically from initial conditions of:

$$m = h = n = p = 0 \quad \text{and} \quad V = -70\,\text{mV}, \tag{36}$$

using the "CNexp" method and a time-step of $25\,\mu\text{s}$. At each time step the injected current I_{e} is drawn from a normal distribution with mean $0.5\,\text{nA}$ and variance

σ^2. The duration of the simulation is 100 ms and the voltage is recorded, which generates a timeseries of 4001 voltage recordings.

Our inference setup follows Lueckmann et al. [43]. There are 12 parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_{12})$ to infer, defined as:

$$\begin{aligned} \theta_1 &= \log(g_\ell) & \theta_7 &= \log(-E_\mathrm{K}) \\ \theta_2 &= \log(\overline{g}_\mathrm{Na}) & \theta_8 &= \log(-V_\mathrm{T}) \\ \theta_3 &= \log(\overline{g}_\mathrm{K}) & \theta_9 &= \log(k_{\beta\mathrm{n}1}) \\ \theta_4 &= \log(\overline{g}_\mathrm{M}) & \theta_{10} &= \log(k_{\beta\mathrm{n}2}) \\ \theta_5 &= \log(-E_\ell) & \theta_{11} &= \log(\tau_\mathrm{max}) \\ \theta_6 &= \log(E_\mathrm{Na}) & \theta_{12} &= \log(\sigma). \end{aligned} \tag{37}$$

The data $\mathbf{x}$ is taken to be 18 features of the voltage timeseries $V(t)$, in particular:

- The mean and log standard deviation of $V(t)$.
- The normalized 3rd, 5th and 7th moments of $V(t)$.
- The logs of the normalized 4th, 6th and 8th moments of $V(t)$.
- The autocorrelation coefficients of $V(t)$ at lags $k \times 2.5$ ms for $k = 1, \ldots, 10$.

The features are linearly transformed to have approximately zero mean and unit covariance matrix; the parameters of the transformation are calculated based on a pilot run. The ground truth parameters $\boldsymbol{\theta}^*$ are taken to be:

$$\begin{aligned} \theta_1^* &= \log\left(10^{-4}\right) & \theta_7^* &= \log(100) \\ \theta_2^* &= \log(0.2) & \theta_8^* &= \log(60) \\ \theta_3^* &= \log(0.05) & \theta_9^* &= \log(0.5) \\ \theta_4^* &= \log\left(7 \times 10^{-5}\right) & \theta_{10}^* &= \log(40) \\ \theta_5^* &= \log(70) & \theta_{11}^* &= \log(1000) \\ \theta_6^* &= \log(50) & \theta_{12}^* &= \log(1). \end{aligned} \tag{38}$$

The prior over parameters is:

$$\theta_i \sim \mathcal{U}(\theta_i^* - \log 2,\ \theta_i^* + \log 1.5) \quad \text{for} \quad i = 1, \ldots, 12. \tag{39}$$

Finally, the observed data $\mathbf{x}_o$ were generated by simulating the model at $\boldsymbol{\theta}^*$.

B Full experimental results

In this section, we include the full set of experimental results. For each simulator model, we report:

- The approximate posterior computed by SNL.
- The trade-off between accuracy and simulation cost. This is reported for all methods.
- The full results of the simulation-based calibration test, consisting of one histogram per parameter.
- The distance-based convergence diagnostic, i.e. the distance between simulated and observed data vs the number of rounds. This is reported for SNL, SNPE-A and SNPE-B.
- The goodness-of-fit diagnostic, i.e. the Maximum Mean Discrepancy between simulated data and data generated from the likelihood model, for a given parameter value (we use the true parameters $\boldsymbol{\theta}^*$). We report this for SNL, NL and a baseline Gaussian fit.

B.1 A toy problem with complex posterior

Figure 5 shows the results. The exact posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x}_o)$ is plotted in Figure 5a. Even though the prior is uniform and the likelihood is Gaussian, the posterior is complex and non-trivial: it has four symmetric modes due to the two squaring operations in Equations (10) and (11), and vertical cut-offs due to the hard constraints imposed by the prior. We can see that the SNL posterior (Figure 5b) approximates the exact posterior well.

B.2 M/G/1 queue model

Figure 6 shows the results. The SNL posterior is shown in Figure 6a. We can see that the posterior is concentrated around the true parameters. Parameter θ_1 is particularly well constrained. From the description of the model in Equations (18)–(20), it directly follows that:

$$\theta_1 \leq \min_i (d_i - d_{i-1}). \tag{40}$$

The data $\mathbf{x}$ is an invertible linear transformation of the quantiles of $\{d_i - d_{i-1}\}_{1:I}$ including the $0th$ quantile, which is precisely equal to $\min_i (d_i - d_{i-1})$. Hence, the data $\mathbf{x}$ imposes a hard constraint on the maximum possible value of θ_1, as correctly captured by the SNL posterior. On the other hand, the data is less informative about θ_2 and θ_3, hence the parameters are less constrained.

B.3 Lotka–Volterra population model

Figure 7 shows the results. The SNL posterior is shown in Figure 7a; the extent of each plot corresponds to the broad prior $p_\mathrm{broad}(\boldsymbol{\theta})$. We can see that the posterior is tightly concentrated around the true parameters, which suggests that the data $\mathbf{x}$ is highly informative about the parameters $\boldsymbol{\theta}$.

B.4 Hodgkin–Huxley cortical pyramidal neuron model

The results are shown in Figure 8. The SNL posterior is shown in Figure 9; the extent of each plot corresponds to the uniform prior. In the SNL posterior, it can be seen that all parameters are clustered around their true values (red dots and lines).

The sodium and equilibrium potentials are relatively tightly clustered, and the potassium equilibrium potential less so:

$$E_{\mathrm{Na}} = 50 \pm 2 \text{ mV} \tag{41}$$

$$E_{\mathrm{K}} = -99 \text{ mV (range } [-121\,\mathrm{mV},\ -90\,\mathrm{mV}]) \tag{42}$$

$$E_{\ell} = -70 \pm 4 \text{ mV}. \tag{43}$$

The tight clustering reflects that concentrations, and hence equilibrium potentials, are maintained within a range by neuronal ion exchangers and and glial buffering [72]. Furthermore, when regulation of ion concentration fails, pathological brain states can arise [72]. The longer tail of the potassium equilibrium potential posterior might be due to it having relatively little influence on the mean of $V(t)$, since at lower potentials the potassium conductance $\overline{g}_{\mathrm{K}} n^4$ will be relatively small, so, according to Equation (31), the potassium current will also be small. The quantity V_{T}, which adjusts the threshold of spike initiation, is also fairly tightly controlled, which will tend to keep the firing rate around a constant value.

In contrast to the equilibrium potentials, the conductances vary more, within a factor of 1.8 for g_{ℓ}, and a factor of 3 for $\overline{g}_{\mathrm{Na}}$, $\overline{g}_{\mathrm{K}}$ and $\overline{g}_{\mathrm{M}}$. Moreover, $\overline{g}_{\mathrm{Na}}$ and $\overline{g}_{\mathrm{K}}$ are correlated, which is consistent with their opposing depolarizing and hyperpolarizing influences on the membrane potential. A higher sodium conductance could lead to the cell being hyper-excitable, but this should be counteracted by a greater potassium conductance. This allows for their wide range, and is consistent with the biological evidence for diverse but correlated sets of channel conductances underlying particular activity patterns [25]. In contrast, $\overline{g}_{\mathrm{M}}$ appears to have relatively little influence over the output, and is not correlated with any other parameters.

The parameter $\tau_{\max}$, which also relates to the M-type potassium channel, also has little effect. Further simulations of other neuron types could be undertaken to see if these parameters are generally loosely constrained, which could then lead to experimentally testable predictions about the density and variability of M-type conductances.

The parameters $k_{\beta \mathrm{n}1}$ and $k_{\beta \mathrm{n}2}$ also have relatively wide ranges, and there is a weak correlation between the two. Increasing $k_{\beta \mathrm{n}1}$ effectively increases the half-activation voltage of potassium conductances, and increasing $k_{\beta \mathrm{n}2}$ makes the slope of transition less pronounced. The lack of posterior at high $k_{\beta \mathrm{n}1}$ (high threshold) and low $k_{\beta \mathrm{n}2}$ (sharper transition) might cause the neuron not to repolarize quickly enough, and hence be hyper-excitable.

Finally, we note that the posterior computed by SNL is qualitatively consistent with the posterior reported by Lueckmann et al. [43] in Figure G.2 of their article.

References

[1] S. Agostinelli et al. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3):250–303, 2003.

[2] J. Alsing, B. Wandelt, and S. Feeney. Massive optimal data compression and density estimation for scalable, likelihood-free inference in cosmology. *Monthly Notices of the Royal Astronomical Society*, 477(3):2874–2885, 2018.

[3] R. Bansal and A. Yaron. Risks for the long run: A potential resolution of asset pricing puzzles. *Journal of Finance*, 59(4):1481–1509, 2004.

[4] M. A. Beaumont. Approximate Bayesian computation in evolution and ecology. *Annual Review of Ecology, Evolution, and Systematics*, 41(1):379–406, 2010.

[5] M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate Bayesian computation in population genetics. *Genetics*, 162:2025–2035, 2002.

[6] M. A. Beaumont, J.-M. Cornuet, J.-M. Marin, and C. P. Robert. Adaptive approximate Bayesian computation. *Biometrika*, 96(4):983–990, 2009.

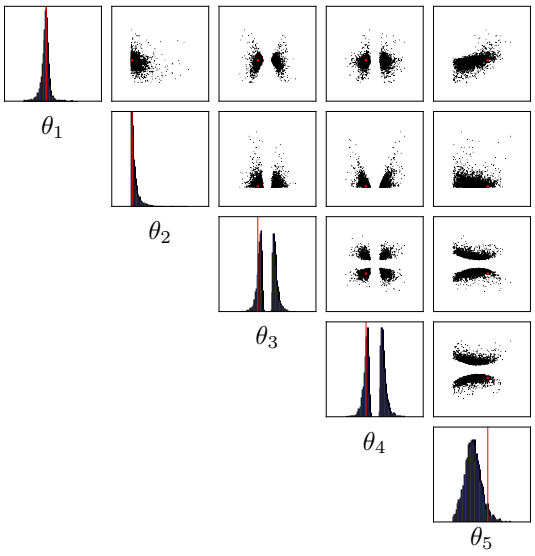
[7] C. M. Bishop. Mixture density networks. Technical Report NCRG/94/004, Aston University, 1994.

[8] M. G. B. Blum. Approximate Bayesian computation: A nonparametric perspective. *Journal of the American Statistical Association*, 105(491): 1178–1187, 2010.

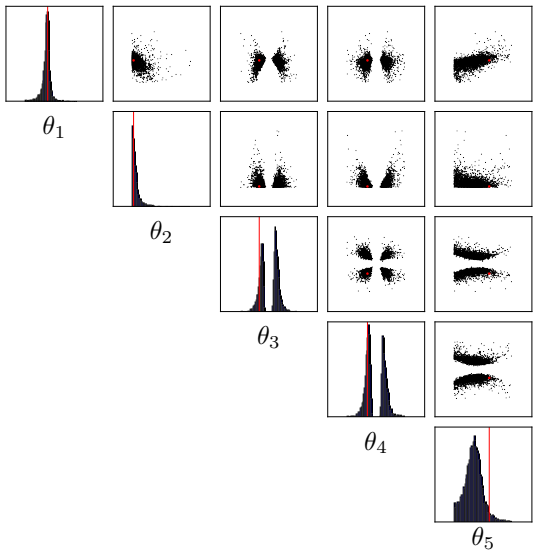
[9] M. G. B. Blum and O. François. Non-linear regression models for approximate Bayesian computation. *Statistics and Computing*, 20(1):63–73, 2010.

[10] F. V. Bonassi and M. West. Sequential Monte Carlo with adaptive weights for approximate Bayesian computation. *Bayesian Analysis*, 10(1): 171–187, 2015.

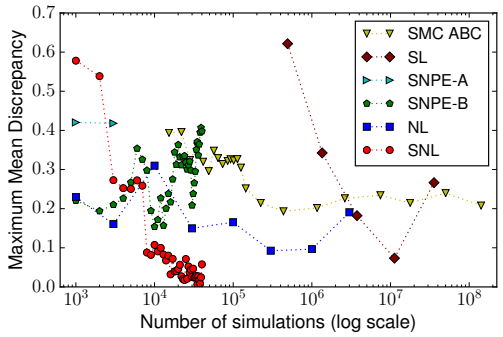
[11] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez. A guide to constraining effective field theories with machine learning. *Physical Review Letters*, 98(5): 052004, 2018.



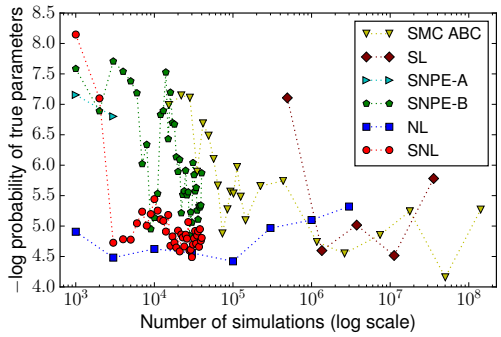
(a) MCMC samples from exact posterior. True parameters indicated in red.



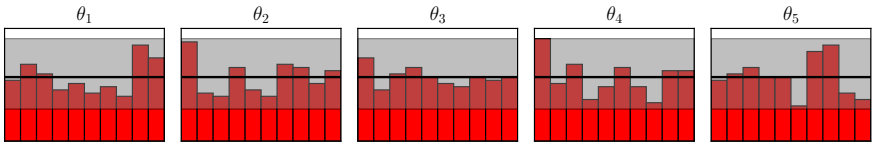
(b) MCMC samples from SNL posterior. True parameters indicated in red.



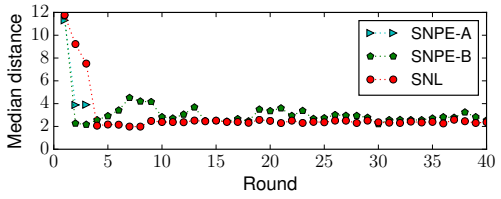
(c) Maximum Mean Discrepancy vs simulation cost.



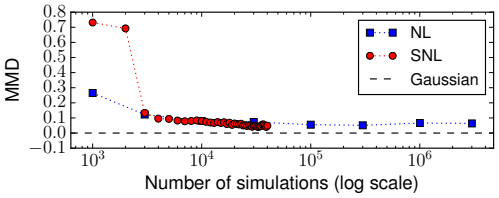
(d) Minus log probability of true parameters vs simulation cost.



(e) Calibration test for SNL. Histogram outside gray band indicates poor calibration.

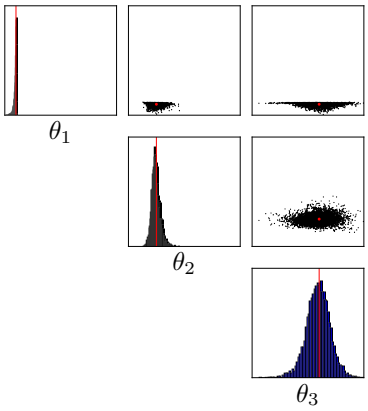


(f) Median distance from simulated to observed data.

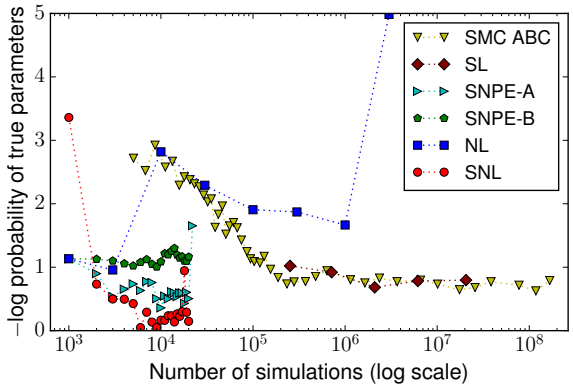


(g) Likelihood goodness-of-fit vs simulation cost, calculated at true parameters.

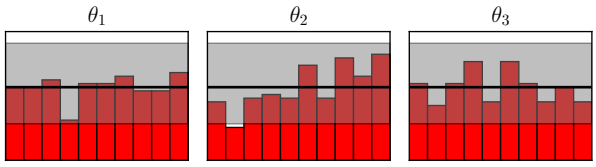
Figure 5: A toy model with complex posterior.



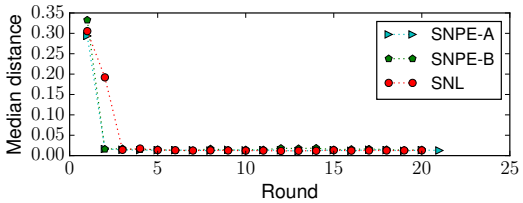
(a) MCMC samples from SNL posterior. True parameters indicated in red.



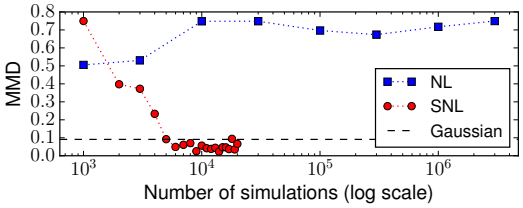
(b) Minus log probability of true parameters vs simulation cost.



(c) Calibration test for SNL. Histogram outside gray band indicates poor calibration.

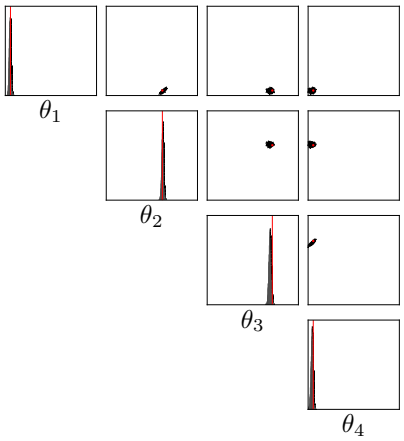


(d) Median distance from simulated to observed data.

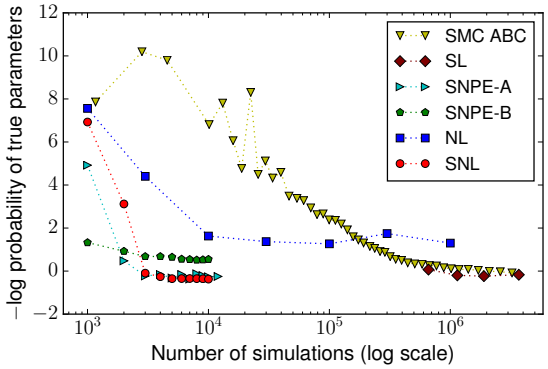


(e) Likelihood goodness-of-fit vs simulation cost, calculated at true parameters.

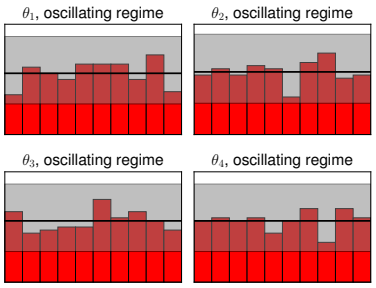
Figure 6: M/G/1 queue model.



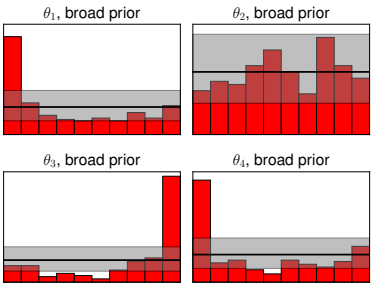
(a) MCMC samples from SNL posterior. True parameters indicated in red.



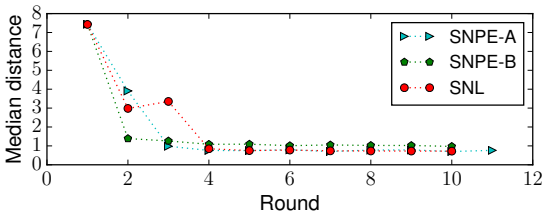
(b) Minus log probability of true parameters vs simulation cost.



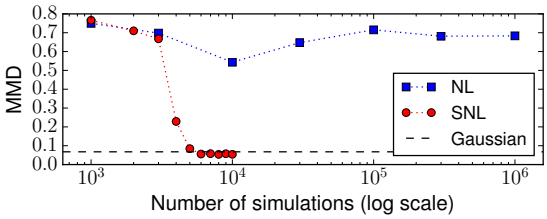
(c) Calibration test for SNL, oscillating regime.



(d) Calibration test for SNL, broad prior.

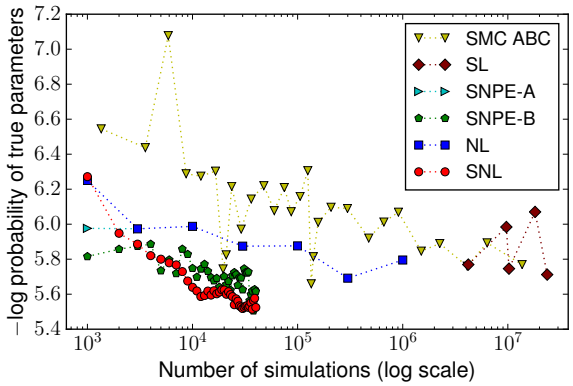


(e) Median distance from simulated to observed data.

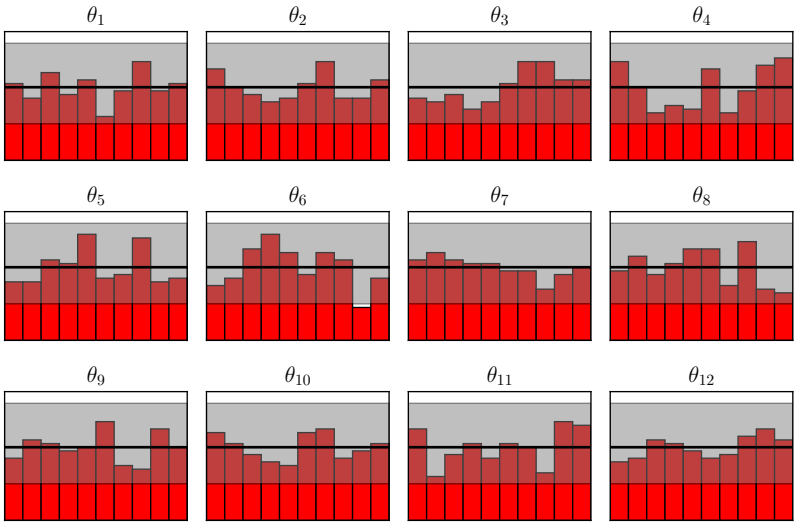


(f) Likelihood goodness-of-fit vs simulation cost, calculated at true parameters.

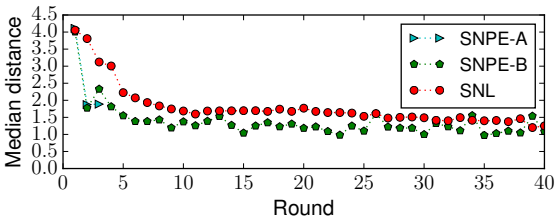
Figure 7: Lotka–Volterra population model.



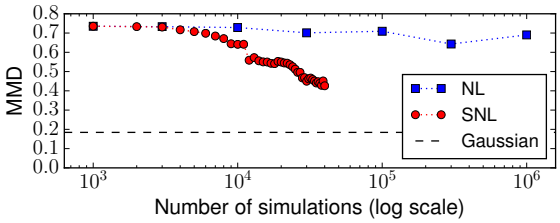
(a) Minus log probability of true parameters vs simulation cost.



(b) Calibration test for SNL. Histogram outside gray band indicates poor calibration.



(c) Median distance from simulated to observed data.



(d) Likelihood goodness-of-fit vs simulation cost, calculated at true parameters.

Figure 8: Hodgkin–Huxley neuron model.

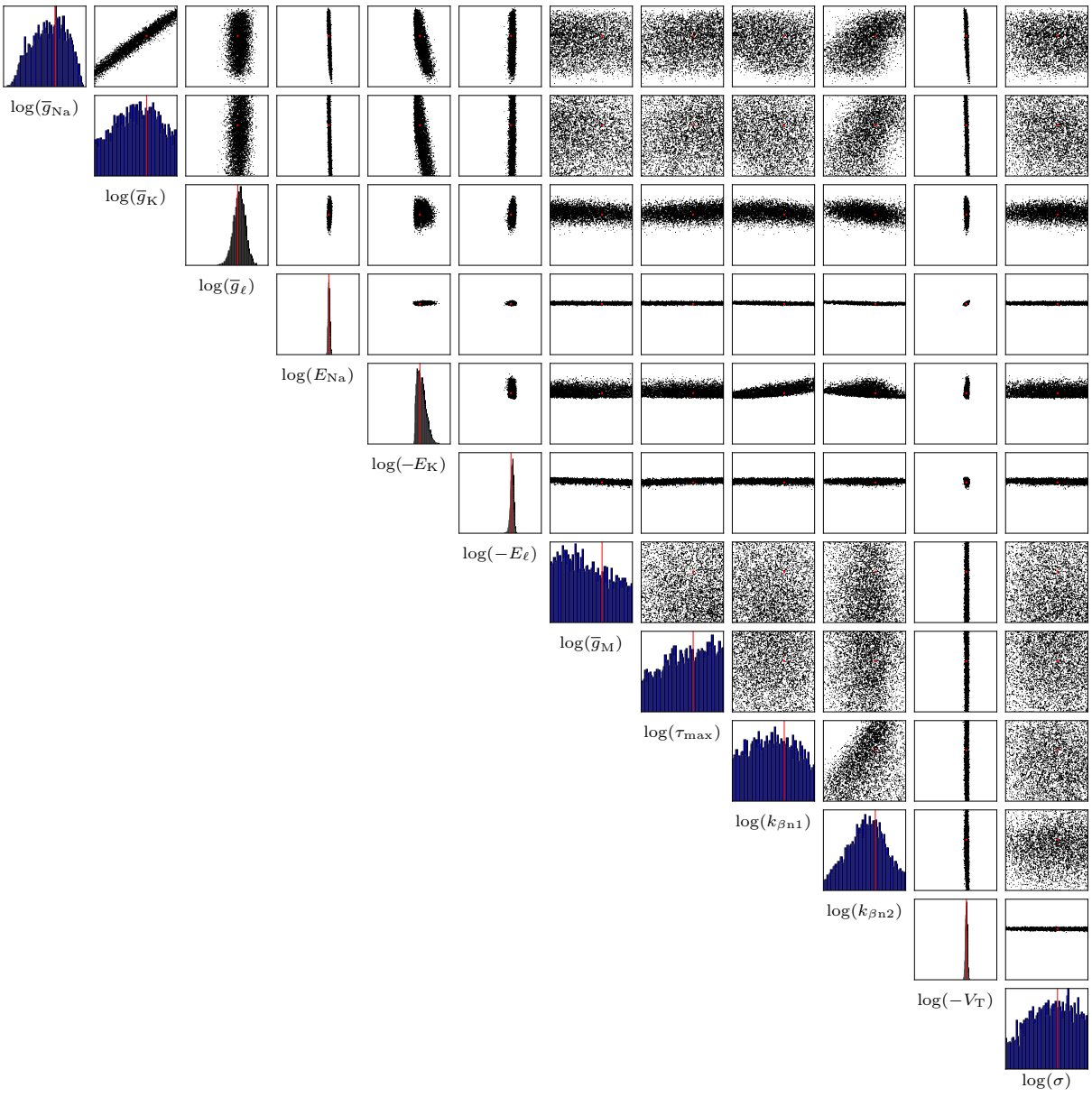


Figure 9: Hodgkin–Huxley model: MCMC samples from SNL posterior. True parameters $\boldsymbol{\theta}^*$ are indicated in red. The range of each histogram is $[\theta_i^* - \log 2,\ \theta_i^* + \log 1.5]$ for $i = 1, \ldots, 12$. Compare with Figure G.2 by Lueckmann et al. [43].

[12] J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer. Mining gold from implicit models to improve likelihood-free inference. *arXiv:1805.12244*, 2018.

[13] T. Carnevale and M. Hines. *The NEURON Book*. Cambridge University Press, 2006.

[14] M. L. Casado, A. G. Baydin, D. M. Rubio, T. A. Le, F. Wood, L. Heinrich, G. Louppe, K. Cranmer, K. Ng, W. Bhimji, and Prabhat. Improvements to inference compilation for probabilistic programming in large-scale scientific simulators. *arXiv:1712.07901*, 2017.

[15] K. Cranmer, J. Pavez, and G. Louppe. Approximating likelihood ratios with calibrated discriminative classifiers. *arXiv:1506.02169*, 2016.

[16] M. F. Cusumano-Towner, A. Radul, D. Wingate, and V. K. Mansinghka. Probabilistic programs for inferring the goals of autonomous agents. *arXiv:1704.04977*, 2017.

[17] A. C. Daly, D. J. Gavaghan, C. Holmes, and J. Cooper. Hodgkin–Huxley revisited: reparametrization and identifiability analysis of the classic action potential model with approximate Bayesian methods. *Royal Society Open Science*, 2(12), 2015.

[18] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

[19] R. Dutta, J. Corander, S. Kaski, and M. U. Gutmann. Likelihood-free inference by ratio estimation. *arXiv:1611.10242*, 2016.

[20] R. G. Everitt. Bootstrapped synthetic likelihood. *arXiv:1711.05825*, 2018.

[21] Y. Fan, D. J. Nott, and S. A. Sisson. Approximate Bayesian Computation via regression density estimation. *Stat*, 2(1):34–48, 2013.

[22] M. Fasiolo, S. N. Wood, F. Hartig, and M. V. Bravington. An extended empirical saddlepoint approximation for intractable likelihoods. *Electronic Journal of Statistics*, 12(1):1544–1578, 2018.

[23] M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked autoencoder for distribution estimation. *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[24] D. T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

[25] M. S. Goldman, J. Golowasch, E. Marder, and L. F. Abbott. Global structure, robustness, and modulation of neuronal models. *The Journal of Neuroscience*, 21(14):5229–5238, 2001.

[26] C. Gouriéroux, A. Monfort, and E. Renault. Indirect inference. *Journal of Applied Econometrics*, 8(S1):S85–S118, 1993.

[27] M. M. Graham and A. J. Storkey. Asymptotically exact inference in differentiable generative models. *Electronic Journal of Statistics*, 11(2):5105–5164, 2017.

[28] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13 (1):723–773, 2012.

[29] M. U. Gutmann and J. Corander. Bayesian optimization for likelihood-free inference of simulator-based statistical models. *Journal of Machine Learning Research*, 17(125):1–47, 2016.

[30] M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander. Likelihood-free inference via classification. *Statistics and Computing*, 28(2):411–425, 2018.

[31] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of Physiology*, 117:500–544, 1952.

[32] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

[33] R. Izbicki, A. Lee, and C. Schafer. High-dimensional density ratio estimation with extensions to approximate likelihood computation. *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, 2014.

[34] M. Järvenpää, M. U. Gutmann, A. Pleska, A. Vehtari, and P. Marttinen. Efficient acquisition rules for model-based approximate Bayesian computation. *Bayesian Analysis*, 2018.

[35] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

[36] D. P. Kingma, T. Salimans, and M. Welling. Variational dropout and the local reparameterization trick. *Advances in Neural Information Processing Systems 28*, 2015.

[37] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka. Picture: A probabilistic programming language for scene perception. *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[38] T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming. *Proceedings of the 20th International*

Conference on Artificial Intelligence and Statistics, pages 1338–1348, 2017.

[39] C. Leuenberger and D. Wegmann. Bayesian computation and model selection without likelihoods. *Genetics*, 184(1):243–252, 2010.

[40] J. Li, D. J. Nott, Y. Fan, and S. A. Sisson. Extending approximate Bayesian computation methods to high dimensions via a Gaussian copula model. *Computational Statistics & Data Analysis*, 106(C): 77–89, 2017.

[41] J. Lintusaari, M. U. Gutmann, S. Kaski, and J. Corander. On the identifiability of transmission dynamic models for infectious diseases. *Genetics*, 202(3):911–918, 2016.

[42] J. Lintusaari, M. U. Gutmann, R. Dutta, S. Kaski, and J. Corander. Fundamentals and recent developments in approximate Bayesian computation. *Systematic Biology*, 66(1):e66–e82, 2017.

[43] J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke. Flexible statistical inference for mechanistic models of neural dynamics. *Advances in Neural Information Processing Systems 30*, 2017.

[44] J.-M. Lueckmann, G. Bassetto, T. Karaletsos, and J. H. Macke. Likelihood-free inference with emulator networks. *arXiv:1805.09294*, 2018.

[45] V. K. Mansinghka, T. D. Kulkarni, Y. N. Perov, and J. B. Tenenbaum. Approximate Bayesian image interpretation using generative probabilistic graphics programs. *Advances in Neural Information Processing Systems 26*, 2013.

[46] J.-M. Marin, P. Pudlo, C. P. Robert, and R. J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

[47] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, 2003.

[48] H. Markram et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163:456–492, 2015.

[49] E. Meeds and M. Welling. GPS-ABC: Gaussian process surrogate approximate Bayesian computation. *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, 2014.

[50] E. Meeds and M. Welling. Optimization Monte Carlo: Efficient and embarrassingly parallel likelihood-free inference. *Advances in Neural Information Processing Systems 28*, 2015.

[51] E. Meeds, R. Leenders, and M. Welling. Hamiltonian ABC. *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, 2015.

[52] A. Moreno, T. Adel, E. Meeds, J. M. Rehg, and M. Welling. Automatic variational ABC. *arXiv:1606.08549*, 2016.

[53] Q. Morris. Recognition networks for approximate inference in BN20 networks. *Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence*, 2001.

[54] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

[55] R. M. Neal. Slice sampling. *The Annals of Statistics*, 31(3):705–767, 2003.

[56] R. M. Neal. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, chapter 5, pages 113–162. Chapman & Hall/CRC., 2011.

[57] V. M. H. Ong, D. J. Nott, M.-N. Tran, S. A. Sisson, and C. C. Drovandi. Variational Bayes with synthetic likelihood. *Statistics and Computing*, 28 (4):971–988, 2018.

[58] B. Paige and F. Wood. Inference networks for sequential Monte Carlo in graphical models. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[59] G. Papamakarios and I. Murray. Fast ϵ-free inference of simulation models with Bayesian conditional density estimation. *Advances in Neural Information Processing Systems 29*, 2016.

[60] G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems 30*, 2017.

[61] K. C. Pham, D. J. Nott, and S. Chaudhuri. A note on approximating ABC-MCMC using flexible classifiers. *Stat*, 3(1):218–227, 2014.

[62] M. Pospischil, M. Toledo-Rodriguez, C. Monier, Z. Piwkowska, T. Bal, Y. Frégnac, H. Markram, and A. Destexhe. Minimal Hodgkin–Huxley type models for different classes of cortical and thalamic neurons. *Biological Cybernetics*, 99(4):427–441, 2008.

[63] L. F. Price, C. C. Drovandi, A. Lee, and D. J. Nott. Bayesian synthetic likelihood. *Journal of Computational and Graphical Statistics*, 27(1):1–11, 2018.

[64] J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman. Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular Biology and Evolution*, 16(12):1791–1798, 1999.

[65] O. Ratmann, O. Jørgensen, T. Hinkley, M. Stumpf, S. Richardson, and C. Wiuf. Using likelihood-free inference to compare evolutionary dynamics of the protein networks of H. pylori and P. falciparum. *PLoS Computational Biology*, 3(11):2266–2278, 2007.

[66] L. Romaszko, C. K. Williams, P. Moreno, and P. Kohli. Vision-as-inverse-graphics: Obtaining a rich 3D explanation of a scene from a single image. *IEEE International Conference on Computer Vision Workshop*, 2017.

[67] C. M. Schafer and P. E. Freeman. Likelihood-free inference in cosmology: Potential for the estimation of luminosity functions. *Statistical Challenges in Modern Astronomy V*, pages 3–19, 2012.

[68] A. Y. Shestopaloff and R. M. Neal. On Bayesian inference for the M/G/1 queue with efficient MCMC sampling. *arXiv:1401.5548*, 2014.

[69] S. A. Sisson and Y. Fan. Likelihood-free Markov chain Monte Carlo. In *Handbook of Markov Chain Monte Carlo*, chapter 12, pages 313–336. Chapman & Hall/CRC., 2011.

[70] S. A. Sisson, Y. Fan, and M. M. Tanaka. Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007.

[71] T. Sjöstrand, S. Mrenna, and P. Skands. A brief introduction to PYTHIA 8.1. *Computer Physics Communications*, 178(11):852–867, 2008.

[72] G. G. Somjen. Ion regulation in the brain: implications for pathophysiology. *The Neuroscientist*, 8(3):254–267, 2002.

[73] D. C. Sterratt, B. Graham, A. Gillies, and D. Willshaw. *Principles of Computational Modelling in Neuroscience*. Cambridge University Press, 2011.

[74] M. Stoye, J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer. Likelihood-free inference with an improved cross-entropy estimator. *arXiv:1808.00973*, 2018.

[75] S. Talts, M. Betancourt, D. Simpson, A. Vehtari, and A. Gelman. Validating Bayesian inference algorithms with simulation-based calibration. *arXiv:1804.06788*, 2018.

[76] T. Toni, D. Welch, N. Strelkowa, A. Ipsen, and M. P. H. Stumpf. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of The Royal Society Interface*, 6(31):187–202, 2009.

[77] D. Tran, R. Ranganath, and D. Blei. Hierarchical implicit models and likelihood-free variational inference. *Advances in Neural Information Processing Systems 30*, 2017.

[78] M.-N. Tran, D. J. Nott, and R. Kohn. Variational Bayes with intractable likelihood. *Journal of Computational and Graphical Statistics*, 26(4):873–882, 2017.

[79] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016.

[80] A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with PixelCNN decoders. *Advances in Neural Information Processing Systems 29*, 2016.

[81] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

[82] A. van den Oord et al. Parallel WaveNet: Fast high-fidelity speech synthesis. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

[83] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305, 2008.

[84] D. J. Wilkinson. *Stochastic Modelling for Systems Biology, Second Edition*. Taylor & Francis, 2011.

[85] S. N. Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466 (7310):1102–1104, 2010.

[86] J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. B. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. *Advances in Neural Information Processing Systems 28*, 2015.

[87] A. Yuille and D. Kersten. Vision as Bayesian inference: analysis by synthesis? *Trends in Cognitive Sciences*, 10(7):301–308, 2006.

4.2 Discussion

In this section, I further discuss the paper *Sequential Neural Likelihood: Fast Likelihood-free Inference with Autoregressive Flows*, which was presented in the previous section. In what follows, I evaluate the contribution and impact the paper has had so far, and I compare Sequential Neural Likelihood with an alternative approach for estimating the simulator's intractable likelihood based on active learning.

4.2.1 Contribution and impact

The paper builds upon our previous work on *Sequential Neural Posterior Estimation (Type A)* and *Masked Autoregressive Flow*. Its main contribution is to identify the limitations of SNPE-A and SNPE-B, and to introduce a new algorithm, *Sequential Neural Likelihood*, that overcomes these limitations. Compared to SNPE, SNL has the following advantages:

(i) SNL is more general, since it can be used with any neural density estimator and any prior distribution. In contrast, SNPE-A can only be used with mixture-density networks and exponential-family priors.

(ii) SNL makes more efficient use of simulations, as it reuses simulated data from previous rounds to train the neural density estimator. In contrast, SNPE only uses the simulations from the last round for training.

(iii) SNL is more robust, as it doesn't need to correct for proposing parameter samples from a distribution other than the prior. As a result, it doesn't suffer from early termination due to negative-definite covariances (such as SNPE-A) or from high variance due to importance weights (such as SNPE-B).

A disadvantage of SNL is that it requires an additional inference step to generate posterior samples. The paper uses MCMC in the form of axis-aligned slice sampling; however, in principle any likelihood-based inference method that can generate samples (such as variational inference) can be used instead. In contrast, SNPE trains a model of the posterior directly, which can generate independent samples without requiring an additional inference step.

The paper suggests using SNL with Masked Autoregressive Flow. MAF is a flexible model for general-purpose density estimation, which makes it a good default choice. However, I should emphasize that SNL is not tied to a particular model, and it can be used with any neural density estimator. In fact, even though this thesis has focused on continuous parameter and data spaces, SNL can in principle be used even when the parameters and/or the data are discrete.

It is too early to evaluate the impact of the paper, as it was published shortly before this thesis was written. As of April 2019, the paper has received 7 citations according to Google Scholar. Along with similar methods, SNL has so far been used for likelihood-free inference in cosmology (Alsing et al., 2019). It remains to be seen whether SNL will prove useful in other applications, and whether it will inspire further research on the topic.

4.2.2 Comparison with active-learning methods

In each round, SNL selects parameters to simulate at by independently proposing from the posterior estimate obtained in the round before. The same strategy is used in SNPE, and a similar strategy is used in SMC-ABC. As we have seen, proposing parameters from the previous posterior estimate can reduce the number of simulations by orders of magnitude compared to proposing parameters from the prior, and can improve posterior-estimation accuracy significantly.

Nonetheless, this strategy was motivated by intuition and empirical validation, rather than a theoretically justified argument. Therefore, it is reasonable to ask whether we can find a more principled strategy that can reduce the number of simulations even further.

In contrast with estimating the posterior, estimating the likelihood can easily accommodate alternative parameter-acquisition strategies. As we saw in the previous chapter, when we estimate the posterior with a neural density model it is necessary to correct for the parameter-acquisition strategy, otherwise the posterior estimate will be biased. On the other hand, as we discussed in section 3 of the paper, the parameter-acquisition strategy does not bias the estimation of the likelihood by a neural density model. Hence, estimating the likelihood instead of the posterior offers increased flexibility in selecting parameters, which enables more sophisticated strategies to be considered.

In principle, we can formulate the problem of selecting parameters as a task of decision-making under uncertainty. Ideally, we would like to select the next parameters such that the resulting simulation gives us the most information about what the posterior should be; such a scheme would select the next simulation optimally in an information-theoretic sense. In the next few paragraphs, I describe a precise theoretical formulation of the above idea.

Suppose that the conditional density of data given parameters is modelled by a Bayesian neural density estimator $q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})$. Given a dataset of simulation results $\mathcal{D} = \{(\boldsymbol{\theta}_1, \mathbf{x}_1), \dots, (\boldsymbol{\theta}_N, \mathbf{x}_N)\}$, our beliefs about $\boldsymbol{\phi}$ are encoded by a distribution $p(\boldsymbol{\phi} \,|\, \mathcal{D})$. Under this Bayesian model, the predictive distribution of $\mathbf{x}$ given $\boldsymbol{\theta}$ is:

$$p(\mathbf{x} \,|\, \boldsymbol{\theta}, \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\phi} \,|\, \mathcal{D})}(q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})), \tag{4.1}$$

whereas the predictive distribution of $\boldsymbol{\theta}$ given $\mathbf{x}$ is:

$$p(\boldsymbol{\theta} \,|\, \mathbf{x}, \mathcal{D}) = \mathbb{E}_{p(\boldsymbol{\phi} \,|\, \mathcal{D})}\left(\frac{q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\int q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}')\, \mathrm{d}\boldsymbol{\theta}'} \right). \tag{4.2}$$

Given observed data $\mathbf{x}_o$, the distribution $p(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D})$ is the *predictive posterior* under the Bayesian model. The predictive posterior encodes two distinct kinds of uncertainty:

(i) Our irreducible uncertainty about what parameters might have generated $\mathbf{x}_o$, due to the fact that the simulator is stochastic.

(ii) Our uncertainty about what the posterior should be, due to not having seen enough simulation results. This type of uncertainty can be reduced by simulating more data.

Our goal is to select the next parameters to simulate at, such that the second kind of uncertainty is maximally reduced.

Now, suppose we obtain a new simulation result $(\boldsymbol{\theta}', \mathbf{x}')$ and hence an updated dataset $\mathcal{D}' = \mathcal{D} \cup \{(\boldsymbol{\theta}', \mathbf{x}')\}$. In that case, our beliefs about $\boldsymbol{\phi}$ will be updated by Bayes' rule:

$$p(\boldsymbol{\phi} \,|\, \mathcal{D}') \propto q_{\boldsymbol{\phi}}(\mathbf{x}' \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\phi} \,|\, \mathcal{D}), \tag{4.3}$$

and the predictive posterior will be updated to use $p(\boldsymbol{\phi} \,|\, \mathcal{D}')$ instead of $p(\boldsymbol{\phi} \,|\, \mathcal{D})$. We want to select parameters $\boldsymbol{\theta}'$ to simulate at such that, on average, the *updated* predictive posterior $p(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D}')$ becomes as certain as possible. We can quantify how uncertain the updated predictive posterior is by its entropy:

$$H(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D}') = -\mathbb{E}_{p(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D}')}(\log p(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D}')). \tag{4.4}$$

The larger the entropy, the more uncertain the predictive posterior is. Hence, under the above framework, the optimal parameters $\boldsymbol{\theta}^*$ to simulate at will be those that minimize the expected predictive-posterior entropy:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}'} \mathbb{E}_{p(\mathbf{x}' \,|\, \boldsymbol{\theta}', \mathcal{D})}(H(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D}')). \tag{4.5}$$

The above expectation is taken with respect to our predictive distribution of $\mathbf{x}'$ given $\boldsymbol{\theta}'$, and not with respect to the actual simulator. Intuitively, this can be though of as running the simulator in our imagination rather than in reality, hence we must take into account our uncertainty over the outcome $\mathbf{x}'$ due to not knowing the simulator's likelihood exactly.

We can justify the above strategy in terms of the mutual information between $\boldsymbol{\theta}$ (the parameters that might have generated $\mathbf{x}_o$) and $\mathbf{x}'$ (the data generated by simulating at $\boldsymbol{\theta}'$). This mutual information can be written as:

$$MI(\boldsymbol{\theta}, \mathbf{x}' \,|\, \boldsymbol{\theta}', \mathbf{x}_o, \mathcal{D}) = H(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D}) - \mathbb{E}_{p(\mathbf{x}' \,|\, \boldsymbol{\theta}', \mathcal{D})}(H(\boldsymbol{\theta} \,|\, \mathbf{x}_o, \mathcal{D}')). \tag{4.6}$$

As we can see, the mutual information between $\boldsymbol{\theta}$ and $\mathbf{x}'$ is equal to the expected reduction in the predictive-posterior entropy due to simulating at $\boldsymbol{\theta}'$. Hence, selecting the next parameters by minimizing the updated expected predictive-posterior entropy is equivalent to selecting the next parameters such that the simulated data gives us on average the most information about what the parameters that produced the observed data might be. The above strategy is a special case of the framework of Järvenpää et al. (2018), if we take the loss function that appears in their framework to be the entropy of the updated predictive posterior.

Although optimal in an information-theoretic sense, the above strategy is intractable, as it involves a number of high-dimensional integrals and the global optimization of an objective. In order to implement the above method, we would need to:

(i) train a Bayesian neural density model,

(ii) compute the predictive distribution of $\mathbf{x}$ given $\boldsymbol{\theta}$ under the model,

(iii) compute the predictive posterior and its entropy, and

(iv) globally minimize the expected predictive-posterior entropy with respect to the next parameters to simulate at.

With current methods, each of the above steps can only be partially or approximately solved. It wouldn't be inaccurate to say that the above strategy reduces the problem of likelihood-free inference to an even harder problem.

Despite being intractable, the above strategy is useful as a guiding principle for the development of other parameter-acquisition strategies, which may approximate the optimal strategy to a certain extent. One such heuristic is the *MaxVar rule* (Järvenpää et al., 2018; Lueckmann et al., 2018); given a Bayesian neural density estimator $q_{\boldsymbol{\phi}}(\mathbf{x} \,|\, \boldsymbol{\theta})$, the MaxVar rule uses the variance of the *unnormalized* posterior density at parameters $\boldsymbol{\theta}'$ (which is due to the uncertainty about $\boldsymbol{\phi}$) as a proxy for the information we would gain by simulating at $\boldsymbol{\theta}'$. According to the MaxVar rule, we select the next parameter to simulate at by maximizing the above variance, that is:

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}'} \mathbb{V}_{p(\boldsymbol{\phi} \,|\, \mathcal{D})}(q_{\boldsymbol{\phi}}(\mathbf{x}_o \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}')). \tag{4.7}$$

Assuming we can (approximately) generate samples $\{\boldsymbol{\phi}_1, \ldots, \boldsymbol{\phi}_M\}$ from $p(\boldsymbol{\phi} \,|\, \mathcal{D})$ (using e.g. MCMC or variational inference), we can approximate the above variance using the empirical variance on the samples as follows:

$$\mathbb{V}_{p(\boldsymbol{\phi} \,|\, \mathcal{D})}(q_{\boldsymbol{\phi}}(\mathbf{x}_o \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}')) \approx \frac{1}{M} \sum_m (q_{\boldsymbol{\phi}_m}(\mathbf{x}_o \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}'))^2 - \left(\frac{1}{M} \sum_m q_{\boldsymbol{\phi}_m}(\mathbf{x}_o \,|\, \boldsymbol{\theta}')\, p(\boldsymbol{\theta}') \right)^2. \tag{4.8}$$

The above empirical variance is differentiable with respect to $\boldsymbol{\theta}'$, so it can be maximized with gradient-based methods and automatic differentiation.

Following the publication of SNL, Conor Durkan, Iain Murray and I co-authored a paper (Durkan et al., 2018) in which we compared the parameter-acquisition strategy of SNL and SNPE with

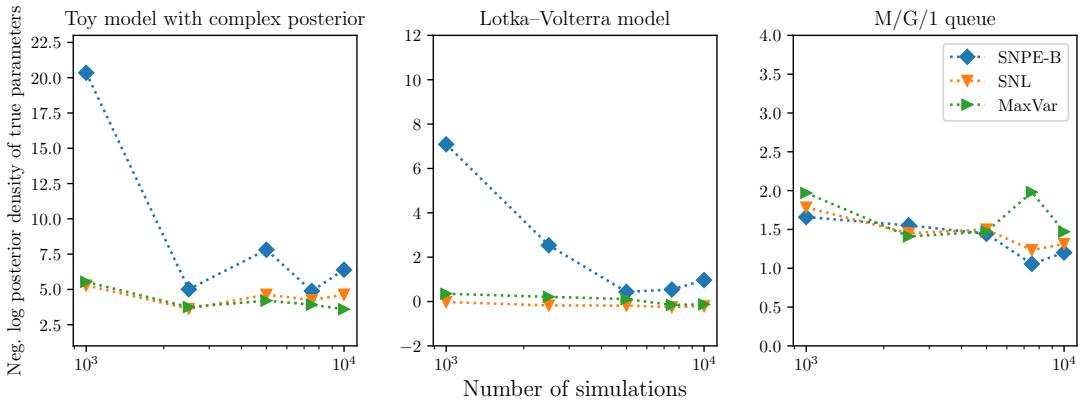


Figure 4.1: Comparison between SNPE-B, SNL and MaxVar for three different simulator models. The plot is taken from (Durkan et al., 2018).

the MaxVar rule described above. The paper, titled *Sequential Neural Methods for Likelihood-free Inference* was presented in December 2018 at the *Bayesian Deep Learning Workshop*, held at the conference *Advances in Neural Information Processing Systems (NeurIPS)*. Conor Durkan was the leading author: he performed the experiments and wrote the paper. Iain Murray and I served as advisors.

The paper compares SNPE-B, SNL and MaxVar. In the experiments, all three algorithms use the same Bayesian neural density estimator, namely a mixture-density network trained with variational dropout, which is the same as that used in the previous chapter. Figure 4.1, reproduced from the original paper with Conor Durkan's permission, shows the negative log posterior density of the true parameters versus the number of simulations. As we can see, despite being to a certain extent theoretically motivated, MaxVar doesn't improve upon SNL. Both SNL and MaxVar improve upon SNPE-B, which agrees with the rest of our experiments on SNL that were presented in this chapter.

In addition to the tradeoff between accuracy and simulation cost, it is worth looking at the wall-clock time of each algorithm. Table 4.1 shows the wall-clock time (in hours) of SNL and MaxVar for each simulator model, using a total of 10^4 simulations. As we can see, SNL is about 10 times faster than MaxVar. The reason for MaxVar being significantly slower than SNL is that MaxVar needs to solve an optimization problem for every parameter it selects. In contrast, SNL proposes parameters via MCMC, which is more efficient.

Table 4.1: Wall-clock time (in hours) per experiment with 10^4 simulations. The table is reproduced from (Durkan et al., 2018).

	Toy model	Lotka–Volterra	M/G/1 queue
SNL	7.48	8.27	7.83
MaxVar	91.73	89.39	70.16

The above comparison shows that SNL takes about the same number of simulations to achieve a given accuracy as a more sophisticated method that is based on active learning. Yet, SNL is an order of magnitude faster in terms of wall-clock time than the active-learning alternative. This comparison doesn't mean that active-learning methods aren't worthwhile in general; the more expensive a simulator is to run, the more important the parameter-acquisition strategy

becomes. However, I would argue that this comparison highlights the importance of evaluating the cost of the parameter-acquisition strategy against the extent to which it makes a better use of simulations. After all, time spent on optimizing for the next parameters to simulate at may also be spent on running more simulations or training a better model.

4.3 Summary and conclusions

Simulators are useful modelling tools, because they are interpretable, flexible, and a good match to our understanding of mechanistic processes in the physical world. However, the likelihood of a simulator model is often intractable, which makes traditional Bayesian inference over the model's parameters challenging. In this and the previous chapter, I discussed likelihood-free inference, i.e. Bayesian inference based on simulations, and I introduced efficient likelihood-free inference methods based on neural density estimation.

Approximate Bayesian computation is a family of methods that have been traditionally used for likelihood-free inference. ABC methods are based on repeatedly simulating the model, and rejecting simulations that don't match the observed data. ABC methods don't target the exact posterior but an approximate posterior, obtained by an alternative observation that the simulated data is at most a distance ϵ away from the observed data. The parameter ϵ trades off efficiency for accuracy; ABC becomes exact as ϵ approaches zero, but at the same time the simulation cost increases dramatically.

In the previous chapter, I presented a method for likelihood-free inference that was later given the name *Sequential Neural Posterior Estimation (Type A)*. SNPE-A trains a Bayesian mixture-density network on simulated data in order to estimate the exact posterior. The algorithm progresses over multiple rounds; in each round, parameters to simulate at are proposed by the posterior estimate obtained in the round before. Unlike ABC, SNPE-A doesn't reject simulations, and doesn't involve setting a distance ϵ. Experiments showed that SNPE-A achieves orders of magnitude improvement over ABC methods such as rejection ABC, MCMC-ABC and SMC-ABC.

SNPE-A has two drawbacks: it is tied to mixture-density networks, and it may terminate early due to the posterior estimate becoming improper. A variant of SNPE-A, which I refer to as SNPE-B, was proposed by Lueckmann et al. (2017) to overcome these limitations. However, as we demonstrated in this chapter, SNPE-B is typically less accurate than SNPE-A (at least when SNPE-A doesn't terminate early) due to increased variance in the posterior estimate.

To overcome the limitations of both SNPE-A and SNPE-B, this chapter presents *Sequential Neural Likelihood*, a method that is similar to SNPE but targets the likelihood instead of the posterior. In our experiments, SNL was more accurate and more robust than both SNPE-A and SNPE-B. In addition, SNL enables the use of diagnostics such as a two-sample goodness-of-fit test between the simulator and the neural density estimator. A comparison between SNL and an active-learning method for selecting parameters, performed by Durkan et al. (2018), showed that SNL is at least as simulation-efficient as the active-learning method, but significantly faster in terms of wall-clock time.

As I discussed in the previous chapter, when using ABC it is often necessary to transform data into lower-dimensional summary statistics, in order to maintain the acceptance rate at a high enough level. One limitation of this thesis is that it didn't explore how the methods that are based on neural density estimation scale with the dimensionality of the parameters or the data. Nonetheless, research in deep learning has demonstrated that neural networks with suitable architectures scale well to high-dimensional inputs or outputs. It is reasonable to expect that, with suitable neural architectures, SNPE or SNL may scale better than ABC to high-dimensional parameters or data.

Some preliminary work in this direction already exists. For example, Lueckmann et al. (2017) used a recurrent neural network with SNPE in order to estimate the posterior directly from timeseries data. Similarly, Chan et al. (2018) used an exchangeable neural network to estimate the posterior when the data is a set of exchangeable (e.g. independent) datapoints. As I argued in the chapter on Masked Autoregressive Flow, building invariances in our neural architectures that reflect reasonable assumptions about the data is a key element in scaling neural networks to high dimensions. As far as I'm aware, a careful evaluation of how neural likelihood-free methods such as SNPE and SNL scale to high dimensions hasn't been performed yet. However, given the progress of deep-learning research, I would argue that engineering neural architectures to use with SNPE and SNL, or alternative neural methods, is a promising direction for future work.

Having explored in the last two chapters neural methods both for estimating the posterior and for estimating the likelihood, it is natural to ask which approach is better. Despite the success of SNL over SNPE, I would argue that there is no definite answer to this question, as each approach has its own strengths and weaknesses. Ultimately the right approach depends on what the user is interested in. Hence, I would argue that it is worthwhile to continue exploring both approaches, as well as improving our current methods and developing new ones.

Bibliography

M. Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`.

S. Agostinelli et al. Geant4—a simulation toolkit. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 506(3): 250–303, 2003.

R. Al-Rfou et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv:1605.02688*, 2016.

A. A. Alemi, B. Poole, I. Fischer, J. V. Dillon, R. A. Saurous, and K. Murphy. Fixing a broken ELBO. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

J. Alsing, B. D. Wandelt, and S. M. Feeney. Massive optimal data compression and density estimation for scalable, likelihood-free inference in cosmology. *Monthly Notices of the Royal Astronomical Society*, 477(3):2874–2885, 2018a.

J. Alsing, B. D. Wandelt, and S. M. Feeney. Optimal proposals for approximate Bayesian computation. *arXiv:1808.06040*, 2018b.

J. Alsing, T. Charnock, S. M. Feeney, and B. D. Wandelt. Fast likelihood-free cosmology with neural density estimators and active learning. *arXiv:1903.00007*, 2019.

C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2):697–725, 2009.

M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. *Proceedings of the 34th International Conference on Machine Learning*, 2017.

R. Basri and D. W. Jacobs. Lambertian reflectance and linear subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):218–233, 2003.

M. Bauer and A. Mnih. Resampled priors for variational autoencoders. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

M. A. Beaumont. Approximate Bayesian computation in evolution and ecology. *Annual Review of Ecology, Evolution, and Systematics*, 41(1):379–406, 2010.

M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate Bayesian computation in population genetics. *Genetics*, 162:2025–2035, 2002.

M. A. Beaumont, J.-M. Cornuet, J.-M. Marin, and C. P. Robert. Adaptive approximate Bayesian computation. *Biometrika*, 96(4):983–990, 2009.

J. Behrmann, W. Grathwohl, R. T. Q. Chen, D. K. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. *arXiv:1811.00995*, 2018.

P. Billingsley. *Probability and Measure*. Wiley, 1995.

C. M. Bishop. *Pattern recognition and machine learning*. Springer New York, 2006.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

M. G. B. Blum and O. François. Non-linear regression models for approximate Bayesian computation. *Statistics and Computing*, 20(1):63–73, 2010.

M. G. B. Blum, M. A. Nunes, D. Prangle, and S. A. Sisson. A comparative review of dimension reduction methods in approximate Bayesian computation. *Statistical Science*, 28(2):189–208, 2013.

F. V. Bonassi and M. West. Sequential Monte Carlo with adaptive weights for approximate Bayesian computation. *Bayesian Analysis*, 10(1):171–187, 2015.

L. Bornn, N. S. Pillai, A. Smith, and D. Woodard. The use of a single pseudo-sample in approximate Bayesian computation. *Statistics and Computing*, 27(3):583–590, 2017.

L. Bottou. Stochastic gradient descent tricks. *Neural Networks, Tricks of the Trade, Reloaded*, 7700:430–445, 2012.

J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez. Constraining effective field theories with machine learning. *Physical Review Letters*, 121(11):111801, 2018a.

J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez. A guide to constraining effective field theories with machine learning. *Physical Review Letters*, 98(5):052004, 2018b.

J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer. Mining gold from implicit models to improve likelihood-free inference. *arXiv:1805.12244*, 2018c.

A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. *Proceedings of the 7th International Conference on Learning Representations*, 2019.

M. Brubaker, M. Salzmann, and R. Urtasun. A family of MCMC methods on implicitly defined manifolds. *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, 2012.

L. Buesing, T. Weber, S. Racanière, S. M. A. Eslami, D. J. Rezende, D. P. Reichert, F. Viola, F. Besse, K. Gregor, D. Hassabis, and D. Wierstra. Learning and querying fast generative models for reinforcement learning. *arXiv:1802.03006*, 2018.

O. Cappé and E. Moulines. Online EM algorithm for latent data models. *Journal of the Royal Statistical Society, Series B*, 71(3):593–613, 2008.

A. Caticha. Relative entropy and inductive inference. *AIP Conference Proceedings*, 707(1):75–96, 2004.

J. Chan, V. Perrone, J. Spence, P. Jenkins, S. Mathieson, and Y. Song. A likelihood-free inference framework for population genetic data using exchangeable neural networks. *Advances in Neural Information Processing Systems 31*, 2018.

T. Charnock, G. Lavaux, and B. D. Wandelt. Automatic physical inference with information maximizing neural networks. *Physical Review D*, 97(8), 2018.

R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems 31*, 2018.

Y. Chen and M. U. Gutmann. Adaptive Gaussian copula ABC. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

H. Choi, E. Jang, and A. A. Alemi. WAIC, but why? Generative ensembles for robust anomaly detection. *arXiv:1810.01392*, 2019.

K. Chwialkowski, H. Strathmann, and A. Gretton. A kernel test of goodness of fit. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

M. F. Cusumano-Towner, A. Radul, D. Wingate, and V. K. Mansinghka. Probabilistic programs for inferring the goals of autonomous agents. *arXiv:1704.04977*, 2017.

I. Danihelka, B. Lakshminarayanan, B. Uria, D. Wierstra, and P. Dayan. Comparison of maximum likelihood and GAN-based training of Real NVPs. *arXiv:1705.05263*, 2017.

N. De Cao, I. Titov, and W. Aziz. Block neural autoregressive flow. *arXiv:1904.04676*, 2019.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv:1810.04805*, 2018.

J. V. Dillon, I. Langmore, D. Tran, E. Brevdo, S. Vasudevan, D. Moore, B. Patton, A. A. Alemi, M. D. Hoffman, and R. A. Saurous. TensorFlow distributions. *arXiv:1711.10604*, 2017.

L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011.

C. Durkan, G. Papamakarios, and I. Murray. Sequential neural methods for likelihood-free inference. *Bayesian Deep Learning Workshop at Neural Information Processing Systems*, 2018.

G. K. Dziugaite, D. M. Roy, and Z. Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *Proceedings of the 31st Conference on Uncertainty in Artificial Intelligence*, 2015.

H. Edwards and A. J. Storkey. Towards a neural statistician. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability and its Applications*, 14(1):153–158, 1969.

S. M. A. Eslami, N. Heess, T. Weber, Y. Tassa, D. Szepesvari, K. Kavukcuoglu, and G. E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. *Advances in Neural Information Processing Systems 29*, 2016.

Y. Fan, D. J. Nott, and S. A. Sisson. Approximate Bayesian computation via regression density estimation. *Stat*, 2(1):34–48, 2013.

P. Fearnhead and D. Prangle. Constructing summary statistics for approximate Bayesian computation: Semi-automatic approximate Bayesian computation. *Journal of the Royal Statistical Society: Series B*, 74(3):419–474, 2012.

M. Germain, K. Gregor, I. Murray, and H. Larochelle. MADE: Masked autoencoder for distribution estimation. *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in Neural Information Processing Systems 27*, 2014.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

M. M. Graham and A. J. Storkey. Asymptotically exact inference in differentiable generative models. *Electronic Journal of Statistics*, 11(2):5105–5164, 2017.

W. Grathwohl, R. T. Q. Chen, J. Betterncourt, I. Sutskever, and D. K. Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. *Proceedings of the 7th International Conference on Learning Representations*, 2018.

K. Gregor, G. Papamakarios, F. Besse, L. Buesing, and T. Weber. Temporal difference variational auto-encoder. *Proceedings of the 7th International Conference on Learning Representations*, 2019.

A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *Joural of Machine Learning Research*, 13(1):723–773, 2012.

A. Grover, M. Dhar, and S. Ermon. Flow-GAN: Combining maximum likelihood and adversarial learning in generative models. *Proceedings of 32nd AAAI Conference on Artificial Intelligence*, 2018.

S. Gu, Z. Ghahramani, and R. E. Turner. Neural adaptive sequential Monte Carlo. *Advances in Neural Information Processing Systems 28*, 2015.

M. U. Gutmann and A. Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of Machine Learning Research*, 13:307–361, 2012.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer New York, 2001.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

G. E. Hinton, P. Dayan, and M. Revow. Modeling the manifolds of images of handwritten digits. *IEEE Transactions on Neural Networks*, 8(1):65–74, 1997.

J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *arXiv:1902.00275*, 2019.

E. Hoogeboom, R. van den Berg, and M. Welling. Emerging convolutions for generative normalizing flows. *arXiv:1901.11137*, 2019.

C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville. Neural autoregressive flows. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics—Simulation and Computation*, 19(2):433–450, 1990.

A. Hyvärinen. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6:695–709, 2005.

A. Hyvärinen and P. Pajunen. Nonlinear independent component analysis: Existence and uniqueness results. *Neural Networks*, 12(3):429–439, 1999.

M. Järvenpää, M. U. Gutmann, A. Pleska, A. Vehtari, and P. Marttinen. Efficient acquisition rules for model-based approximate Bayesian computation. *Bayesian Analysis*, 2018.

W. Jitkrittum, W. Xu, Z. Szabo, K. Fukumizu, and A. Gretton. A linear-time kernel goodness-of-fit test. *Advances in Neural Information Processing Systems 30*, 2017.

T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *Proceedings of the 6th International Conference on Learning Representations*, 2018a.

T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. *arXiv:1812.04948*, 2018b.

N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *Proceedings of 5th International Conference on Learning Representations*, 2017.

S. Kim, S. gil Lee, J. Song, and S. Yoon. FloWaveNet: A generative flow for raw audio. *arXiv:1811.02155*, 2018.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference on Learning Representations*, 2015.

D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1×1 convolutions. *Advances in Neural Information Processing Systems 31*, 2018.

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. *Advances in Neural Information Processing Systems 29*, 2016.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25*, 2012.

A. Kucukelbir, R. Ranganath, A. Gelman, and D. M. Blei. Automatic variational inference in Stan. *Advances in Neural Information Processing Systems 28*, 2015.

T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka. Picture: A probabilistic programming language for scene perception. *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

X. Li and W. Grathwohl. Training Glow with constant memory cost. *Bayesian Deep Learning Workshop at Neural Information Processing Systems*, 2018.

Q. Liu, J. D. Lee, and M. Jordan. A kernelized Stein discrepancy for goodness-of-fit tests. *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, 2016.

J.-M. Lueckmann, P. J. Goncalves, G. Bassetto, K. Öcal, M. Nonnenmacher, and J. H. Macke. Flexible statistical inference for mechanistic models of neural dynamics. *Advances in Neural Information Processing Systems 30*, 2017.

J.-M. Lueckmann, G. Bassetto, T. Karaletsos, and J. H. Macke. Likelihood-free inference with emulator networks. *Symposium on Advances in Approximate Bayesian Inference at Neural Information Processing Systems*, 2018.

D. J. C. MacKay. *Information Theory, Inference & Learning Algorithms*. Cambridge University Press, 2002.

V. K. Mansinghka, T. D. Kulkarni, Y. N. Perov, and J. B. Tenenbaum. Approximate Bayesian image interpretation using generative probabilistic graphics programs. *Advances in Neural Information Processing Systems 26*, 2013.

P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré. Markov chain Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 100(26):15324–15328, 2003.

H. Markram et al. Reconstruction and simulation of neocortical microcircuitry. *Cell*, 163:456–492, 2015.

G. J. McLachlan and K. E. Basford. *Mixture models: Inference and applications to clustering*. Marcel Dekker, 1988.

J. Menick and N. Kalchbrenner. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *Proceedings of the 7th International Conference on Learning Representations*, 2019.

T. P. Minka. Expectation propagation for approximate Bayesian inference. *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence*, 2001.

T. Müller, B. McWilliams, F. Rousselle, M. Gross, and J. Novák. Neural importance sampling. *arXiv:1808.03856*, 2018.

I. Murray. *Advances in Markov chain Monte Carlo methods*. PhD thesis, Gatsby computational neuroscience unit, University College London, 2007.

C. Nash and C. Durkan. Autoregressive energy machines. *arXiv:1904.05626*, 2019.

C. Nash, S. M. A. Eslami, C. Burgess, I. Higgins, D. Zoran, T. Weber, and P. Battaglia. The multi-entity variational autoencoder. *Learning Disentangled Features Workshop at Neural Information Processing Systems*, 2017.

R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto, 1993.

S. Nowozin. Effective sample size in importance sampling, Aug. 2015. URL `http://www.nowozin.net/sebastian/blog/effective-sample-size-in-importance-sampling.html`. Published online.

J. Oliva, A. Dubey, M. Zaheer, B. Poczos, R. Salakhutdinov, E. Xing, and J. Schneider. Transformation autoregressive networks. *Proceedings of the 35th International Conference on Machine Learning*, 2018.

B. Paige and F. Wood. Inference networks for sequential Monte Carlo in graphical models. *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

G. Papamakarios. Distilling model knowledge. MSc by Research thesis, School of Informatics, University of Edinburgh, 2015.

G. Papamakarios. Preprocessed datasets for MAF experiments, 2018. URL `https://doi.org/10.5281/zenodo.1161203`.

G. Papamakarios and I. Murray. Distilling intractable generative models. *Probabilistic Integration Workshop at Neural Information Processing Systems*, 2015.

G. Papamakarios and I. Murray. Fast ϵ-free inference of simulation models with Bayesian conditional density estimation. *Advances in Neural Information Processing Systems 29*, 2016.

G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems 30*, 2017.

G. Papamakarios, D. C. Sterratt, and I. Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. *Autodiff Workshop at Neural Information Processing Systems*, 2017.

M. Pospischil, M. Toledo-Rodriguez, C. Monier, Z. Piwkowska, T. Bal, Y. Frégnac, H. Markram, and A. Destexhe. Minimal Hodgkin–Huxley type models for different classes of cortical and thalamic neurons. *Biological Cybernetics*, 99(4):427–441, 2008.

D. Prangle. Adapting the ABC distance function. *Bayesian Analysis*, 12(1):289–309, 2017.

D. Prangle, P. Fearnhead, M. P. Cox, P. J. Biggs, and N. P. French. Semi-automatic selection of summary statistics for ABC model choice. *Statistical applications in genetics and molecular biology*, 13(1):67–82, 2014.

R. Prenger, R. Valle, and B. Catanzaro. WaveGlow: A flow-based generative network for speech synthesis. *arXiv:1811.00002*, 2018.

J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, and M. W. Feldman. Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular Biology and Evolution*, 16(12):1791–1798, 1999.

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12 (1):145–151, 1999.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *Proceedings of the 4th International Conference on Learning Representations*, 2016.

R. Ranganath, S. Gerrish, and D. M. Blei. Black box variational inference. *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics*, 2014.

O. Ratmann, O. Jørgensen, T. Hinkley, M. Stumpf, S. Richardson, and C. Wiuf. Using likelihood-free inference to compare evolutionary dynamics of the protein networks of H. pylori and P. falciparum. *PLoS Computational Biology*, 3(11):2266–2278, 2007.

S. J. Reddi, S. Kale, and S. Kumar. On the convergence of Adam and beyond. *Proceedings of 6th International Conference on Learning Representations*, 2018.

D. J. Rezende. Short notes on divergence measures, July 2018. URL `https://danilorezende.com/wp-content/uploads/2018/07/divergences.pdf`. Published online.

D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *Proceedings of the 32nd International Conference on Machine Learning*, 2015.

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *Proceedings of the 31st International Conference on Machine Learning*, 2014.

L. Romaszko, C. K. I. Williams, P. Moreno, and P. Kohli. Vision-as-inverse-graphics: Obtaining a rich 3D explanation of a scene from a single image. *IEEE International Conference on Computer Vision Workshop*, 2017.

R. Salakhutdinov and I. Murray. On the quantitative analysis of deep belief networks. *Proceedings of the 25th Annual International Conference on Machine Learning*, 2008.

T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. PixelCNN++: Improving the PixelCNN with discretized logistic mixture likelihood and other modifications. *Proceedings of the 5th International Conference on Learning Representations*, 2017.

H. Salman, P. Yadollahpour, T. Fletcher, and K. Batmanghelich. Deep diffeomorphic normalizing flows. *arXiv:1810.03256*, 2018.

C. M. Schafer and P. E. Freeman. Likelihood-free inference in cosmology: Potential for the estimation of luminosity functions. *Statistical Challenges in Modern Astronomy V*, pages 3–19, 2012.

Y. Schroecker, M. Vecerik, and J. Scholz. Generative predecessor models for sample-efficient imitation learning. *Proceedings of 7th International Conference on Learning Representations*, 2019.

D. W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. John Wiley & Sons, 1992.

D. Silver et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529 (7587):484–489, 2016.

B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986.

S. A. Sisson, Y. Fan, and M. M. Tanaka. Sequential Monte Carlo without likelihoods. *Proceedings of the National Academy of Sciences*, 104(6):1760–1765, 2007.

T. Sjöstrand, S. Mrenna, and P. Skands. A brief introduction to PYTHIA 8.1. *Computer Physics Communications*, 178(11):852–867, 2008.

D. C. Sterratt, B. Graham, A. Gillies, and D. Willshaw. *Principles of Computational Modelling in Neuroscience*. Cambridge University Press, 2011.

M. E. Tipping and C. M. Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 21(3):611–622, 1999.

T. Toni, D. Welch, N. Strelkowa, A. Ipsen, and M. P. H. Stumpf. Approximate Bayesian computation scheme for parameter inference and model selection in dynamical systems. *Journal of The Royal Society Interface*, 6(31):187–202, 2009.

D. Tran, M. D. Hoffman, D. Moore, C. Suter, S. Vasudevan, and A. Radul. Simple, distributed, and accelerated probabilistic programming. *Advances in Neural Information Processing Systems 31*, 2018.

R. van den Berg, L. Hasenclever, J. M. Tomczak, and M. Welling. Sylvester normalizing flows for variational inference. *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence*, 2018.

A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu. WaveNet: A generative model for raw audio. *arXiv:1609.03499*, 2016a.

A. van den Oord, N. Kalchbrenner, L. Espeholt, K. Kavukcuoglu, O. Vinyals, and A. Graves. Conditional image generation with PixelCNN decoders. *Advances in Neural Information Processing Systems 29*, 2016b.

S. Vikram, M. D. Hoffman, and M. J. Johnson. The LORACs prior for VAEs: Letting the trees speak for the data. *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Publishing Company, 2010.

M. Welling, C. K. I. Williams, and F. V. Agakov. Extreme components analysis. *Advances in Neural Information Processing Systems 16*, 2004.

D. J. Wilkinson. *Stochastic Modelling for Systems Biology, Second Edition*. Taylor & Francis, 2011.

R. D. Wilkinson. Approximate Bayesian computation (ABC) gives exact results under the assumption of model error. *Statistical applications in genetics and molecular biology*, 12(2): 129–141, 2013.

C. K. I. Williams and F. V. Agakov. Products of Gaussians and probabilistic minor component analysis. *Neural Computation*, 14(5):1169–1182, 2002.

S. N. Wood. Statistical inference for noisy nonlinear ecological dynamic systems. *Nature*, 466 (7310):1102–1104, 2010.

J. Wu, I. Yildirim, J. J. Lim, B. Freeman, and J. B. Tenenbaum. Galileo: Perceiving physical object properties by integrating a physics engine with deep learning. *Advances in Neural Information Processing Systems 28*, 2015.

M. D. Zeiler. ADADELTA: An adaptive learning rate method. *arXiv:1212.5701*, 2012.

L. Zhang, W. E, and L. Wang. Monge–Ampère flow for generative modeling. *arXiv:1809.10188*, 2018.

Z. M. Ziegler and A. M. Rush. Latent normalizing flows for discrete sequences. *arXiv:1901.10548*, 2019.

D. Zoran and Y. Weiss. From learning models of natural image patches to whole image restoration. *Proceedings of the 13rd International Conference on Computer Vision*, 2011.